NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE DESIGN AND IMPLEMENTATION OF A COMPILER FOR THE OBJECT-ORIENTED DATA MANIPULATION LANGUAGE

by

Carlos Martin Barbosa
Aykut Kutlusan

September 1995

Thesis Advisor: David K. Hsiao

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE THE DESIGN AND IMPLEMENTATION OF A COMPILER FOR THE OBJECT-ORIENTED DATA MANIPULATION LANGUAGE	5. FUNDING NUMBERS
6. AUTHOR(S) Barbosa, Carlos M. Kutlusan, Aykut	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Object-oriented relationships, such as inheritance and covering can not be readily incorporated in classic data models. This thesis addresses this problem by the design and implementation of an object-oriented data model (O-ODM), that incorporates the object-oriented paradigm.

A Multimodel and Multilingual Database System called M^{2}DBS has been developed at the Naval Postgraduate School. This system incorporates the classic database data models along with a recently developed object-oriented data model (O-ODM). The problem addressed by this work is to design a new object-oriented data manipulation language (O-ODML) for the O-ODM. The approach is to develop and construct an O-ODML Compiler. Then assimilate the compiler with the Kernel Mapping System (KMS) of the M^{2}DBS.

The result of this thesis is a compiler for the O-ODML of the O-ODM. This O-ODML compiler takes an O-ODM query converts it into a low level intermediate language before translating it into a format that the Real Time Monitor can execute on the M^{2}DBS

14. SUBJECT TERMS Mutimodel and Multilingual Database System, Object-Oriented data model Object-Oriented data manipulation language (O-ODML)	15. NUMBER OF PAGES 187
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THE DESIGN AND IMPLEMENTATION OF A COMPILER FOR THE OBJECT-ORIENTED DATA MANIPULATION LANGUAGE

Carlos Martin Barbosa
Lieutenant, United States Navy
BS, U.S. Naval Academy, 1989
and
Aykut Kutlusan
Lieutenant Junior Grade, Turkish Navy
BS, Turkish Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Authors: [illegible]
Carlos Martin Barbosa

[illegible]
Aykut Kutlusan

Approved by: [illegible]
David K. Hsiao, Thesis Advisor

[illegible]
C. Thomas Wu, Co-Advisor

Ted Lewis
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Object-oriented relationships, such as inheritance and covering can not be readily incorporated in classic data models. This thesis addresses this problem by the design and implementation of an object-oriented data model (O-ODM), that incorporates the object-oriented paradigm.

A Multimodel and Multilingual Database System called M^2DBS has been developed at the Naval Postgraduate School. This system incorporates the classic database data models along with a recently developed object-oriented data model (O-ODM). The problem addressed by this work is to design a new object-oriented data manipulation language (O-ODML) for the O-ODM. The approach is to develop and construct an O-ODML Compiler. Then assimilate the compiler with the Kernel Mapping System (KMS) of the M^2DBS.

The result of this thesis is a compiler for the O-ODML of the O-ODM. This O-ODML compiler takes an O-ODM query converts it into a low level intermediate language before translating it into a format that the Real Time Monitor can execute on the M^2DBS.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUD

The value of data in today's information society requires data to be readily accessible. Furthermore, if data are stored in separate systems, they must be shared among these systems. If data cannot be easily shared among systems due to database design incompatibilities, the cost of database accesses and manipulations increases rapidly. These problems are more prevalent in large and long-standing organizations such as the Department of Defense. The proliferation of database systems and the heterogeneity of various databases have led to a multiplicity of database work, a redundancy of database storage, and a conglomeration of database systems which do not have the ability to readily share data.

Especially with today's fiscal constraints, the Department of Defense must be able to consolidate their resources in man, machine and software in order to save funds and increase productivity. Several solutions have been proposed to solve this problem such as:

- A single and massive conversion to a single database and a single database system
- Maintaining the original databases and database systems and implementing the translation software between any two databases and their systems
- A hybrid of the above two solutions.

All of these solutions suffer from some shortcomings. The first solution would require that all data be stored in a single database design. This would require that all the operators learn a new database system thereby increasing the cost of training operators. Further, the particular database design may not be well suited for storing and manipulating the data in question. In a diverse organization such as the Department of Defense, a single database design for all the diverse databases is not a viable option. The second solution would require the translation software between every pair of heterogeneous databases and their transactions in the organization. For a small number of heterogeneous databases and systems, this is a feasible solution. When the number, n, of databases and database systems is large then the number of different translation software increases at a rate of $O(n^2)$, i.e. in the order of n^2. Therefore, the second solution becomes prohibitively costly. A more reasonable solution would convert all the data to a simple database design while allowing the original users to view their data in their original designs and to process the database with their original transactions. This solution is being explored in the Multimodel / Multilingual Database System M^2DBS. [Ref. 1] M^2DBS contains all the model / language interfaces for various pairs of the data models and data languages are supported on a kernel database system, known as the attribute-based database system.

B. MOTIVATION

The overall goal for this project is to construct an Object-Oriented Interface on the Attribute-Based Database System (ABDS). The construction of this interface is to specifically allow for abstract object-oriented concepts such as objects, inheritance, classes and covering. The data definition language (DDL) directly incorporates these concepts. [Ref 2 and 3.] The interface is to contain a wide range of querying and manipulation capabilities. [Ref. 4] This thesis is to design and implement a compiler to translate Object-oriented queries and manipulations into equivalent ABDS queries and manipulations.

The compilation and translation of object-oriented queries and manipulations into ABDS queries and manipulations involved several steps. First, each query or manipulation must be scanned in order to identify its tokens. To produce the scanner, we used the tool LEX. As the scanner identifies the tokens a parser is needed to check the syntactic and semantic correctness of the query or manipulation. The tool YACC is used to produce the parser which accepts all the tokens and completes the parsing. Next, an intermediate language is developed to simplify the translation process. Finally, there is the query constructor which reads the intermediate-language table, the data dictionary, and the symbol table and writes the actual ABDS queries or statements. This final part of the compiler constitutes the bulk of the work, due to the limited functionality of the existing

ABDS data manipulation language compared to the relatively high level of functionality found in the Object-oriented data manipulation language (O-ODML). Since data being queried or manipulated are not accessible to the compiler, a method of translating individual lines and even parts of each line, must be developed to accommodate for the differences in data structures and language syntax of the two systems. Once the query or manipulation statement is converted and constructed, it is then passed to the real time monitor (RTM) [Ref. 5] for its execution.

C. THESIS ORGANIZATION

This thesis is organized in four more chapters and twelve appendicies. In Chapter II we present the object-oriented constructs and the attribute based constructs that are relevant to the thesis. Since the complete definition of these constructs are the work of other thesis, for an in depth review of the constructs you should refer to the those documents. In addition, this chapter points out the conflicting nature of the two models. Chapter III introduces the reasons for choosing LEX and YACC to serve as development tools for the scanner and parser. Additionally, the manner of incorporating the symbol table and intermediate language table and the implementation of the scanner and lexer into the query constructor is also discussed in this chapter. In Chapter IV the query constructor and the general steps of it's algorithm are presented. Additionally, the real time monitor and the pseudocode which controls its operation are introduced. Also, in this chapter a sample query is translated. In Chapter V we make concluding remarks, state accomplishments and limitations.

The appendicies contain the in depth explanation of all aspects of the thesis. Throughout the work preparing for this thesis a series of documents were prepared to aid in the final writing of the thesis. Virtually every paragraph will refer to the appendices to amplify the text given in the thesis. Appendix A is the header file that contains the constructs of symbol table. Appendix B contains the definitions of the DML tokens and the C code that creates the symbol table. Appendix C contains the definitions of the DML grammer and the calls to the functions of the intermediate language table main file. Appendix D is the rules that explain how to fill in the intermediate language table. Appendix E is the intermediate language table header file which contains the constructs and names of the functions used in building the table. Appendix F is the intermediate language table implementation file. Appendix G is the query constructor header file. Appendix H is the query constructor implementation file. Appendix I is the DML compiler main file which serves as the main file for the entire program. Appendix J is the instruction manual for the DML compiler. Appendix K is the translation algorithm for the conversion and construction process in the query constructor. And finally the pseudocode is defined in Appendix L.

II. OBJECT-ORIENTED AND ATTRIBUTE BASED CONSTRUCTS

A. OBJECT-ORIENTED CONSTRUCTS

The DDL developed for the object-oriented data model (O-ODM) contains the following object-oriented constructs: objects, object identifiers, object classes, inheritance, covering and methods. For a complete description of these constructs and their scope refer to [Ref. 2], [Ref. 3] and [Ref. 4].

Class *Class_name* {
  attribute_type *attribute_name*$_1$;

  attribute_type *attribute_name*$_x$;
  return_type *method_name*$_1$;

  return_type *method_name*$_y$;
};

Figure 1. Generic Object

The genric structure of an O-ODDL object class is shown in Figure 1. This generic example does not demonstrate the flexibilty and power of the object-oriented paradigm in modeling a data base. Figure 2, shows some of the object classes of the Faculty student (FACTSTU) database in which inherit, cover, set_of, and inverse_of relationships are displayed. By examining Figure 2, and comparing it with the reprenstation, of the FACTSU database, displayed in Figure 3, the flexibility of the O-ODM should be apparent.

Operations performed by the DML on the data demonstrate great flexibilty and power as well. The operations do require that the end user understand the object-oriented paradigm, but gives the user a more rational view of the data. The complete description of the DML can be found in [Ref. 4].

```
class Name{
  char_string   fname;
  char          mi;
  char_string   lname;
};

class Address{
  char_string   street;
  char_string   city;
  char          state[2];
  char_string   zipcode;
};

class Person{
  Name     pname;
  Address  paddress;
  char     sex;
}

class Faculty : inherit Person{
  char_string      dept;
  set_of Course    teaches;
};

class Student : inherit Person{
  char_string      student_no;
  char_string      major;
  set_of Course    schedule;
};

class Course{
  char_string          cname;
  char_string          cse_no;
  char_string          sec_no;
  Faculty              instructor;
  inverse_of Student.schedule   roster;
};

class Team : cover Student{
  char_string        prjname;
  set_of Civ_Fac     advisor;
};

class Civ_Fac : inherit Faculty{
  char_string                 title;
  inverse_of Team.advisor     advises;
};
```

Figure 2. Sample Classes from the Faculty Student Database

B. ATTRIBUTE-BASED CONSTRUCTS

The basic building block of the Attribute Based Data Model (ABDM) is the attribute-value pair. The attribute defines the qualities of the value. A set of these attribute

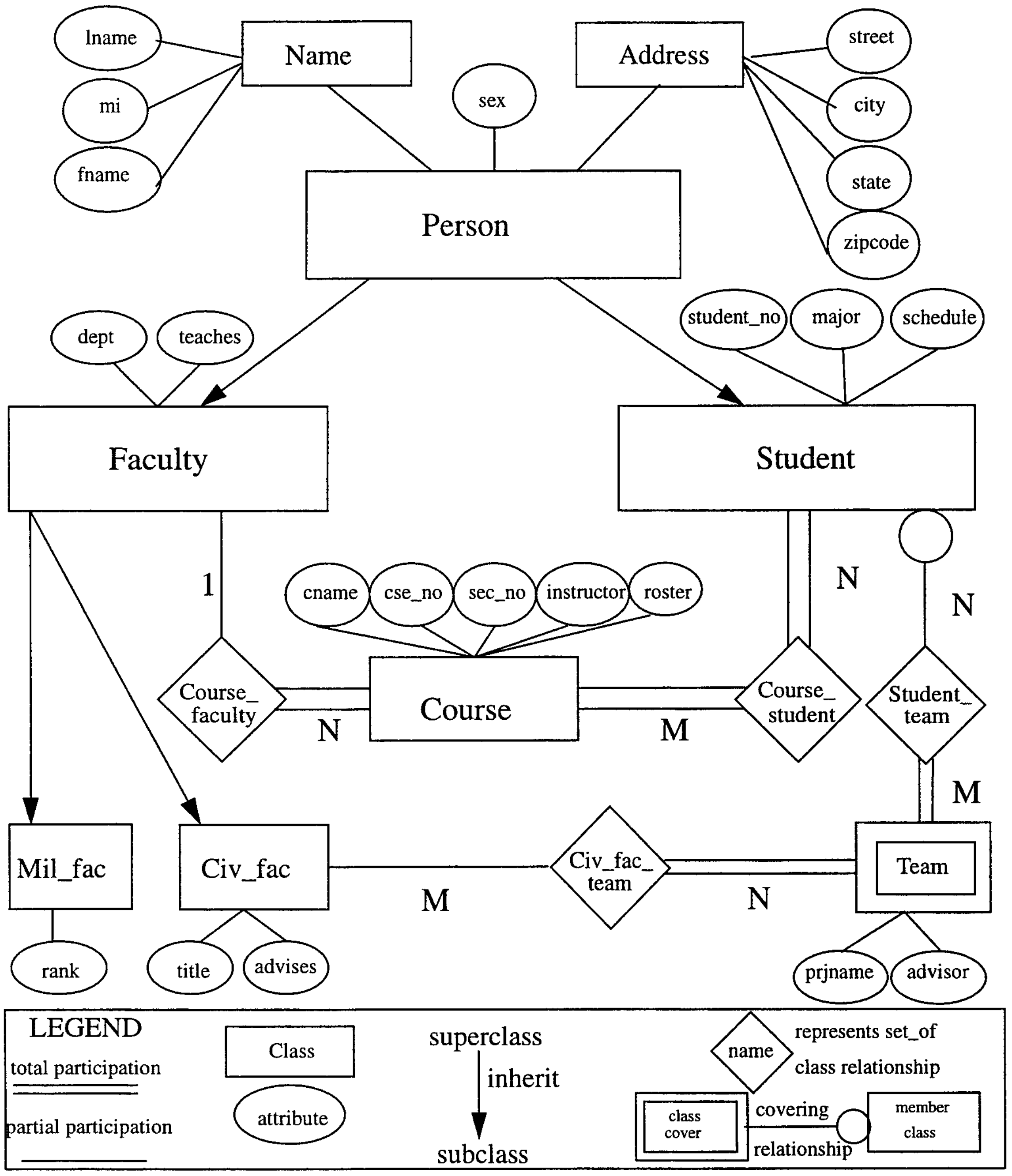


Figure 3. Relatioinship Diagram of the Faculty Student Database

value pairs defines a record, and the records are stored in templates. For a complete description of the ABDM and the available ABDS operations see [Ref. 1].

C. CONFLICTING NATURE OF THE O-ODM AND ABDM

When writing a search statement for the ABDS the user has understood logical ands in the operation. What is not available is "or" statements. Thus requiring searching the same template twice to accomplish what a single O-ODM query can do in a single simple search.

The end user in an object-oriented database when conducting searches, does not have to insure that the attributes he is referring to are directly in the class being queried. As long as dot notation is used to give a path expression signifying a relationship or inheritance with another class. The classic data models require that a search operation work on only one schema or table at a time thus requiring multiple searches with set operations to accomplish the task of one object-oriented search. The ABDS, like conventional models can only search one template at a time; moreover, unlike conventional models it presently can only execute a single operation without passing data between consecutive operations nor are set operations available.

The six operations available on the ABDS can not pass data from one search to another with the exception of the retrieve common operation. However, the retrieve common operation is very limited in its application since it can only pass information between two retrieves and not the remaining operations. This problem is significant in that a query involving multiple templates could not be executed unless the user incrementally queried the ABDS and then manually reentered the answers from the previous retrieve as the argument for the next operation. In other words if the first operation returned 25 answers then the user would have to enter each answer as an argument for the next operation 25 separate times. This presents a significant obstacle in translating a query since the O-ODM contains inheritance, cover, set_of, and inverse_of relations which cause a single O-ODM query to automatically require multiple retrieve statements in the translation into ABDS queries. Due to limited resources specifically time the solution to this problem was to create a pseudocode [see Appendix L] and a real time monitor (RTM) [refer to Ref. 5] that could interpret the pseudocode and execute the queries incrementally.

III. THE COMPILER

A. CHOOSING A SCANNER AND PARSER

Since the task was to build a compiler to translate Object-oriented queries into their Attribute-based equivalent, a compiler was definitely needed. Several problems were considered in deciding the course of action to take. First and foremost was the complexity and magnitude of the scanner and parser. Since the data manipulation language (dml) contained only 59 tokens a scanner would not have been too difficult to build, without the aid of a software tool. However, the dml consisted of over 84 production rules which in itself presented a major challenge in defining, without knowing if the rules were flawed. Additionally the problem of compatibility between the scanner and parser was considered. The probability of developing incompatible software or software that required significant modification in order to communicate was too great in choosing specific scanners and parsers. For these reasons the software tools of LEX and YACC together were used to produce the scanner and parser software. These tools were the logical choice for this task since both tools simplify the task of building a compiler, are guaranteed to be compatible and aid in catching errors in the token definitions and production rules.

B. LEX

LEX is a software tool that allows the user to input token descriptions and then produces the code for the scanner, see [Ref. 6]. The token descriptions are input into LEX and it produces the file *lex.yy.c* which later is compiled with the output of YACC. Additionally, LEX allows the user to chose to execute a routine written in C code, on identification of a token. This code maybe added to the definitions given in the input file.

1. The Symbol Table

The creation of a symbol table was required to maintain a record of each identifier (variable) named in an object oriented query. The information stored in the symbol table is the name, the type and the class name of each variable. The name of each variable is simply the character combination chosen by the user to represent some value. The type of each variable requires a little explanation. Since the object-oriented paradigm refers to objects as the building block of data structures, a logical choice for keys in the OODM would be object identifiers (OID). In order to simplify operations in both data models, it was decided that only OID's would be passed between operations until the final query, at which time the users desired manipulations would be executed. Because of this approach only two types of variables are relevant in this object-oriented model: object references (obj_ref) and object sets (obj_set). An obj_ref is simply a variable which can contain only one OID. While an obj_set is a variable that contains a set of variables which can be zero or many OID's.

C. YACC

YACC is a software tool that allows the user to input a grammar and then produces the code for the parser, see [Ref. 7]. The grammar, also referred to as production rules, is input into YACC and the tokens identified in the LEX specification file must also be identified in the YACC specification file. When the specification file is input into YACC, it generate the C source file *y.tab.c* and *y.tab.h*. It is these two files that are compiled with the LEX output to create the compiler. Similar to LEX, YACC also allows the user to input C code and execute routines upon identification of a production rule.

1. The Intermediate Language Table

The purpose for the intermediate language table was to provide a structure to break down O-ODM queries and store them in a common format to allow for simplification of the translation of a query into the ABDS format. Rules were implemented to fill the intermediate language table according to the operations found in the O-ODM query, see Appendix D.

D. IMPLEMENTATION

In this section, we will discuss the implementation of the scanner and parser in the compilation process. Figure 4, shows how LEX and YACC tie in with the compiler model.

For further information on the use of LEX and YACC see [Ref. 8] . For the O-ODDL compiler implementation of LEX and YACC see [Ref. 2].

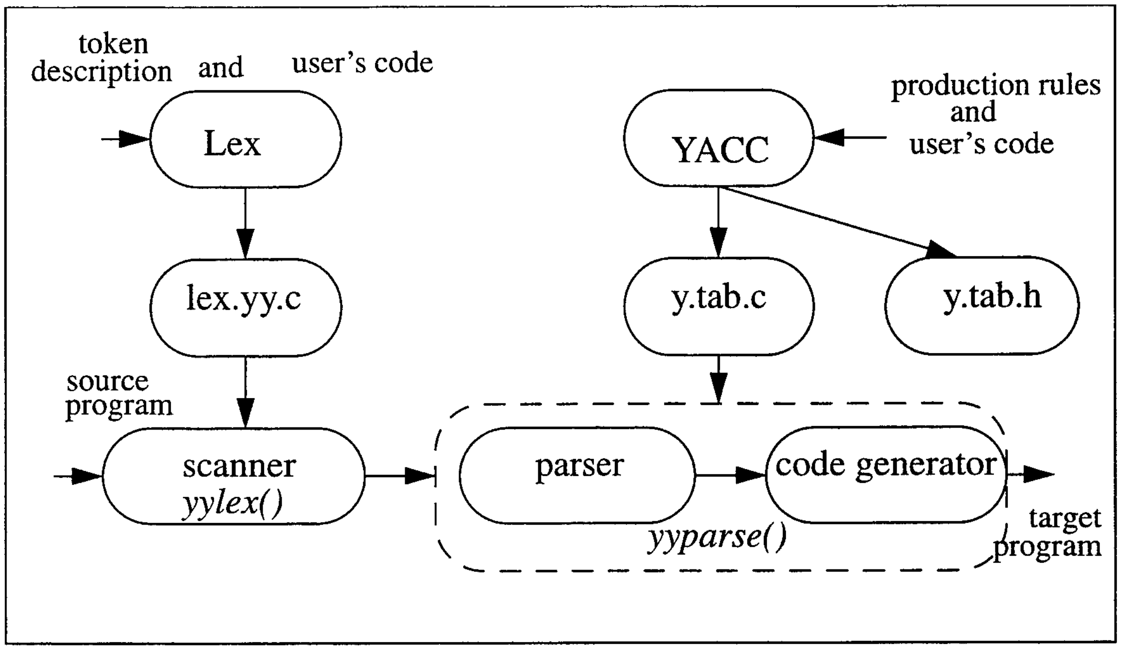


Figure 4. Scanner and Parser Implementation

1. The Scanner

The implementation of the scanner in the compilation process is as a subroutine called by the parser. The scanner takes an arbitrary input stream and tokenizes it, i.e., divides it up into lexical tokens. This tokenized output is then used as the input for the parser generated by YACC. The LEX specification file *dml_lex.l* was written in order to create a set of patterns which the scanner matches against the input see Appendix B. A special C routine named *symtablook()* was also embedded in the specification file to construct the symbol table during the scanning process. Lex itself does not produce an executable program; instead it translates the lex specification into a C source file named *lex.yy.c*. This file contains a C routine called *yylex()* which is a scanner. The executable scanner routine is produced by compiling *lex.yy.c* and linking it with the lex library -ll.

The primary task of the scanner is to recognize the tokens specified in the lex specification. The YACC parser calls *yylex()* to run the scanner. If the scanner encounters an unspecified token while scanning an input query, it gracefully terminates with an invalid token error, otherwise it sends all valid tokens to the parser.

In order to construct the symbol table, the scanner sets a flag for each type of declaration and upon reading the associated variable, it calls the embedded C routine *symtablook()* to construct the symbol table.

2. The Parser

The implementation of the parser in the compilation process is as a subroutine called by the main program of O-ODML compiler. Refer to the Appendix J for a complete discussion of this main program and its interface with the M^{2}DBS. The YACC specification file *dml_yacc.y* contains the specified tokens in the LEX specification file, the production rules and the embedded C routines to construct the intermediate table see Appendix C. YACC takes this specification file and generates the files named *y.tab.h* and y.tab.c. The executable parser subroutine *yyparse()* is produced as a result of compiling the source file *y.tab.c* with a regular C compiler.

When the parser routine *yyparse()* is called by the main program, this routine calls *yylex()* to tokenize the input query. The scanner then reads in the input of characters, until it finds a token of interest to the parser. The parser tries to find a list of tokens which matches a production rule, provided in the YACC specification file. When the parser is done with the parsing process, either it outputs a "successfully parsed message" and calls the query constructor or it terminates gracefully with an error message including the corresponding line number.

In order to construct the intermediate table, the parser traces specific sequences of tokens in the production rules while it is scanning the input query. As soon as the parser has enough tokens to recognize a structure for the intermediate language table then it calls the related C routine defined in the file *dml_yacc.y* thus initiating the intermediate language table construction process.

IV. THE QUERY CONSTRUCTOR

A. GENERATING THE ATTRIBUTE BASED QUERY

The main program of the O-ODML compiler is inside the file *dml_compiler.c* [Appendix I], initiates the call to the parser, which is really how the compilation and translation process is commenced. Once the scanner and parser have completed the symbol table and intermediate language table respectively, the parser calls the function *query_constructor()* in the file *dml_comp.c* [Appendix H] (from this point on will be referred to as the query constructor) which actually does the translating of the queries. Prior to commencing the job of translating the queries the function *createDict()* accomplishes the task of converting the data dictionary file from a text format to a linked list format to simplify the search algorithm for the data dictionary. The query constructor then enters a switch statement (branch statement) in which it reads in the operation block of the intermediate language table and chooses the appropriate function call to convert that operation and its related arguments into an equivalent query. During this conversion the query constructor verifies the declaration of the variables used, to insure the correct type and class name are used. The type checking is accomplished by reading the symbol table. The next phase of the query constructor is to actually construct the ABDS query. This is where the pseudocode and ABDS manipulation langage merge to create a file that is read by the RTM and executed.

B. READING THE INTERMEDIATE LANGUAGE TABLE

Since the intermediate language table is stored in a linked list of records (struct in C), see Appendix E, the method for locating and extracting information is through the use of pointers, loops and conditional statements. Each record in the linked list contains the row number of the table, the operation to be completed, up to two arguments to be used in the operation, and a result block which may contain a variable to be used in the conversion of the query. The operation indicates which function from the file *dml_comp.c*, see Appendix

H, will control the translation of that particular part of the query. The functions operate in two specific steps, the actual conversion phase which involves reading the data dictionary, and the construction phase which builds the operation with known arguments in the ABDS format. The construction phase is fairly self explanatory, but the conversion phase is more complex. Since attributes in the O-ODM can contain dot notation the location of an attribute in the ABDS may not exist in the same class as that of the attributes originally selected in the query. Since the ABDS can search only one template at a time, there exists the requirement to identify within which template the specified attributes are located. This requirement can only be satisfied by reading the data dictionary and extracting the true location (path expression) of the specified attribute.

C. READING THE DATA DICTIONARY

Since the data dictionary is converted from a text file to linked list, the job of searching for specific data was generalized into functions which could be used over again. The first argument to be presented to the dictionary must always be a class name. Since O-ODM queries requires a classname for the desired return type, this is the argument used to enter the data dictionary. The function in_dictionary() takes the argument of the class name and searches the data dictionary for a template with that name and sets the current attribute pointer to the first attribute of that class. The next argument to be presented to the data dictionary is an attribute name. The function in_class() takes the argument of an attribute name and checks to see if that name exists within the current class and sets the pointer to it if found. The next check is to see if the contents of the relationship type column (rel_type) is empty indicating a primitive attribute. Then the construct call for this operation can be made. If the contents of the rel_type column is the string "store", indicating a "set_of" or "inverse_of" relationship, then a check of the attribute type (attr_type) column is made. If the type in the attr_type is a "set_of" then the search starts over in the class that is indicated in the reference table (ref_table) column. If the type in the attr_type column is an "inverse_of" then the contents of the ref_table column must be tokenized in order to continue the search. Another possibility for the contents of the rel_type column is the string "ref" or "cover", if either these two are found then the search simply continues in the class indicated in the ref_table column. If the function in_class() does not find the attribute, it possibly still is related to the current class by way of the inheritance relationship. In that case a function to search for this relationship exist within dml_comp.c see Appendix H. For further explanation as to how the search of data in the data dictionary is accomplished see the convert functions in Appendix F or the algorithm in Appendix K.

D. READING THE SYMBOL TABLE

The search of data in the symbol table is also done through the use of pointers, loops and conditional statements. Specifically, three searches are done in the symbol table. The first is to verify that a variable has been declared. The next is to look for the associated class name of a variable used. And finally, to check to see if the variable type matches the operation. These types of checks for data in the symbol table occurs throughout all the conversion functions defined within the *dml_comp.c* file, but are easier to identify in the algorithm found in Appendix K.

E. THE REAL TIME MONITOR

The specific reason to create the RTM was discussed in Chapter II section C. The functionality of the RTM and the pseudocode developed to pass instructions for data handling and query execution are discussed in Appendix L and [Ref. 5]. There are three parts to the pseudocode that is passed to the RTM and they are the pseudocode, well formed ABDS queries and nearly executable ABDS queries. The pseudocode is simply instructions to the RTM that clarify how operations are to be executed and how they interrelate with other operations, again see Appendix L for specifics. Well formed queries are those that are constructed by the Query constructor and are ready to be executed by the RTM. Nearly executable queries are those that require single or double substitution in order to be executable. These substitutions must be done by the RTM by inserting answers from previous queries.

V. EXECUTION OF A SAMPLE QUERY

In order to demonstrate the complexity of compiling and translating the queries, an example translation involving most of the features of this system follows. The sample translation occurs in three phases, the first being the object oriented query in the newly developed O-ODML see figure. The second phase shows the completed intermediate language table after compilation. Finally, the last phase displays the completed translation with pseudocode, well formed and nearly executable operations.

The O-ODM query, in figure, is one in which the user is attempting to add two courses to a student's schedule. The first two lines of the query declare three variables that will be used in the query, "p" and "i" which are of type object reference and "c" which is of type object set. During compilation these three variables will be entered into the symbol table. The next line of the query is a "find_one" search statement with the class name being that of "Student". The fact that it is a "find_one" and the variable "p" is of type object_ref indicates that only a singular response will be allowed. The following line is a "find_many"

```
Query ChangeStudentSchedule IS
obj_ref  p, i;
obj_set  c;
Begin
p := find_one Student where pname.lname = 'Badgett'
                       and student_no = '20';
c := find_many Course where cse_no = '4114'
                        and sec_no = '1'
                        or cse_no = '3320'
                        and sec_no = '2';
For Each i IN c
  add(p.schedule, i);
End_loop;
End;
```

Figure 5. Sample O-ODML Query

search which includes an "or" as a logical argument. Next is a "for_ loop" which loops once for each member of a given set. Each member of the set is assigned to the same index variable "i" as it enters the loop, therefore each member can than be used in operations within the loop. Inside of the loop an "add" operation is adding each member of the set to the schedule of student "p", one at a time. This is followed by an end of loop statement and an end of query statement.

A. PHASE 1 - Scanning and Parsing

During this phase the parser calls the scanner and passes it the pointer to the input query file. The scanner reads the characters in the query file until it identifies a token to send to the parser. When the scanner encounters a type declaration of an *obj_ref* or *obj_set* it sets the type flag and once the variable is read in it calls the function *symbtablook()* to add the variable name and variable type in the symbol table. Once the scanner reaches the "Begin" in the query file the symbol table has been completed. The scanner continues to pass tokens to the parser which is attempting to match these tokens to the production rules using a top-down search. Once the parser receives the last token from the scanner either it has returned an "unsuccessfully parsed" with a syntax error or it outputs a "successfully parsed" message and calls the query constructor.

B. PHASE 2 - Filling in the Intermediate Language Table

Even though the filling in of the intermediate language table is completed during the parsing phase, it can be thought of as a separte phase in the execution of a query. In figure 8 , the compilation of the O-ODM query is broken down into its most atomic form and placed in the intermediate language table. For the most part the intermediate language table is self explanatory and the rules for filling in the table can be found in Appendix D. What may need explanation is the introduction of new variables in the result column and the word "jump" in the operation column. The new variables that were introduced when the parser filled in the table are those starting with t0 and ending with t5. The purpose of these variables is to allow each line of the table to be treated as the arguments to single simple ABDS operations, as will become evident from examining figure 9. The word jump in the table, indicates a discontinuity in the sequence of execution of the code. Notice that the last entry is not an "End" statement signifying the end of the query as in the O-ODM query. It is not needed since the intermediate language table is stored in a linked list, the fact that the last pointer points to a null value is sufficient to indicate the end of the query.

Line No.	OPERATION	ARGUMENT1	ARGUMENT2	RESULT
0.	find_one	student		p
1.	=	pname.lname	badgett	t0
2.	=	student_no	20	t1
3.	and	t0	t1	0
4.	find_many	course		c
5.	=	cse_no	4114	t0
6.	=	sec_no	1	t1
7.	and	t0	t1	t2
8.	=	cse_no	3320	t3
9.	=	sec_no	2	t4
10.	and	t3	t4	t5
11.	or	t2	t5	0
12.	jump	i	c	
13.	add	p.schedule	i	
14.	end_loop			

Figure 6. Sample Intermediate Language Table

C. PHASE 3 - Constructing the ABDS Query

The completed product out of the query constructor appears just like the text in figure 6 with the exception that the text is stored in a linked list for ease in reading the data sequentially. (The column of numbers on the right hand side of figure 6 were inserted into the illustration to aid the reader in identifying lines of the pseudocode. These numbers do not actually appear in the pseudocode translation passed to the RTM.) The first two lines are for the declaration of *obj_ref* and *obj_set* variables, respectively. The list of two-letter variables starting with the letter "s" or "r" are those variables created by the query constructor itself, which allow the translation to be a series of simple single argument ABDS operations. The "&" signifies that the return values from the next operation are to be stored in the variable following the symbol. The "~" implies that a single substitution must occur in the next operation prior to the execution of that line. The substitution variable is that variable following the "~". In other words that next line is a nearly executable ABDS line. On line nine, two consecutive lines that read as follows are found, indicating the completion of the "find_one" operation:

```
&p
*s2, s3
```

Figure 7. Pseudocode End of O-ODML Find_one Operation

At this point the amount of simple ABDS operations to substitute one O-ODML operation should amplify the power and flexibilty found in the O-ODM. The "find_many" operation is finally answered on lines twenty-four and twenty-five which read as follows:

```
&c
+s6, s9
```

Figure 8. Pseudocode End of O-ODML Find_many Operation

The "*" and "+" are set operations which conduct intersection and union of the OIDs stored within the two-letter variables. A for_loop is indicated by "$" followed by a index variable and the obj_set variable "c" on line twenty-six. On the next line a "#" implies a double substitution warning for the following insert statement. This insert statement is preceeded by an "A" to indicate it is an ADD type insert. Finally the "!" signifies the end of the for_loop. The "?" inside the INSERT statement indicates that the RTM must make a call to the OID generator since this operation will be creating a new record. The OID generator exists within the kernal system. Again an "End" statement is not included in the pseudocode for the same reason as the intermediate language table, the peudocode is stored in a linked list.

```
%i,p                                                        (1)
@c, sa, sb, sc, sd, se, sf, sg, sh, si, sj, sk              (2)
&sa                                                         (3)
[RETRIEVE((TEMP = Name) and (LNAME = badgett)) (OID)]       (4)
&sb                                                         (5)
~sa                                                         (6)
[RETRIEVE((TEMP = Person) and (PNAME = sa)) (OID)]          (7)
&sc                                                         (8)
[RETRIEVE((TEMP = Student) and (STUDENT_NO = 20)) (OID)]    (9)
&p                                                          (10)
*sb, sc                                                     (11)
&se                                                         (12)
[RETRIEVE((TEMP = Course) and (CSE_NO = 4114))(OID)]        (13)
&sf                                                         (14)
[RETRIEVE((TEMP = Course) and (SEC_NO = 1))(OID)]           (15)
&sg                                                         (16)
*se, sf                                                     (17)
&sh                                                         (18)
[RETRIEVE((TEMP = Course) and (CSE_NO = 3320))(OID)]        (19)
&si                                                         (20)
[RETRIEVE((TEMP = Course) and (SEC_NO = 2))(OID)]           (21)
&sj                                                         (22)
*sh, si                                                     (23)
&c                                                          (24)
+sg, sj                                                     (25)
$i, c                                                       (26)
# p, i                                                      (27)
[AINSERT(<TEMP,Course_student>,<OID,?>,
<OID_COURSE,i>,<OID_STUDENT,p>)]                            (28)
!                                                           (29)
```

Figure 9. Pseudocode Translation of O-ODM Query

VI. CONCLUSIONS

A. ACCOMPLISHMENTS

The primary goal of this thesis research was to design and implement a compiler to translate Object-oriented queries and manipulations into equivalent ABDS queries and manipulations. In order to accomplish this task, several steps needed to be completed. First, a scanner and parser had to be chosen to identify the tokens and check for syntactic and semantic correctness. Additionally, the scanner and parser had to be used for the construction of a symbol table and an intermediate language table respectively. Next a query constructor was created to convert and construct the queries in the pseudocode format which the RTM reads and executes. Finally, documentation was provided to explain all the steps in detail and to assist follow on research, see Appendices A-L.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

Due to the scope of this project several operations were not completely implemented in the query constructor and RTM. The O-ODML operations Insert, Read_input, and Delete were not fully implemented due to the defined need for the user to be interactive with the query constructor or the RTM. The RTM is the logical choice for this interaction since, by the time the RTM begins to execute queries the query constructor is no longer operating. However, the RTM is not able to access the data dictionary or the original query, therefore all instructions concerning the interface must come from the query constructor. Therefore a new protocol must now be added to instruct the RTM to interact with the user and not just the ABDS.

The arithmetic functions of count, min, max, and avg were not included because of the time constraint in completing the project. Again the actual arithmetic operations must occur in the RTM, since the data can only be accessed by the RTM. Likewise, the query constructor must provide the instructions to the RTM for conducting the operations.

Also an optimization program can be written to shorten the number of queries entered into the ABDS by combining some of the queries that are logically consecutive and are searching the the same template. This optimization will need to be incorporated in the query constructor once the translation is complete and prior to calling the RTM.

C. SUMMARY

In summary, the O-ODML compiler and interface was constructed and implemented successfully. The inability of the ABDS to search multiple tables simultaneously, pass data between operations and the absence of an "or" operator was overcome by the creation of the RTM. The capability to read a text data dictionary and convert it into the linked list format, in order to use it in aiding the translation of queries, has greatly expanded the capabilities of this model beyond its on data storage in the ABDS. If the data dictionary incorporated in this project is used as the standard for all the models, then the O-ODM will now be able to query any data stored in the ABDS. Though the ABDS may appear limited in its capabilities to serve as the base model for the M^2DBMS it has the ability to operate at the atomic level operations required to support searches, which is what is needed to accomplish the diversity of the M^2DBMS project.

APPENDIX A - SYMBOL TABLE HEADER FILE

```
/***************************************************************************

file name: symtabl.h

purpose :  This file contains the declaration for "symtablook()" and also

           the definition of the "symtabl" structure to include limiting the

           symtable entries to a quantity of 20.

***************************************************************************/



#define SIZE 20 /* maximum number of variables */


struct symtabl {

   char *name;

   char *type;

   char *clsname;

} symtabl[SIZE];


struct symtabl *symtablook();
```

APPENDIX B - DML LEX FILE

```
/*************************************************************************
file name: dml_lex.l
purpose :  This file contains the definitions of the DML tokens which are
           the input to the LEX Unix tool and the source code for
           constructing the symbol table.
*************************************************************************/


%{
#include <stdio.h>
#include <string.h>
#include "dml.tab.h"
#include "symtabl.h"
#include <ctype.h>

/* "define input()" is used to make LEX case insensitive to the input text */
# define input() (((dmltchar=dmlsptr>dmlsbuf?U(*--
dmlsptr):tolower(getc(dmlin)))==10?(dmllineno++,dmltchar):dmltchar)==EOF?0:dm
ltchar)

/* "sym_flag" stores the name of the variable type. */
int sym_flag;
symflag = 0;

%}
%%
```

```
/* The following lines contain token definitions. */

[ \t\n]*        {/* skip whitespace */}

"//".*           ;   /* comment to the end of a line */

add             { return(ADD);}

and             { yylval.t_str = strdup(yytext);return(LOGICAL_OPERATOR);}

avg             { return(AVG);}

begin           { return(BEGIN_Q);}

char            { return(CHAR);}

char_string     { sym_flag = 1; return(CHAR_STRING);}

class           { return(CLASS);}

contains        { return(CONTAINS);}

count           { return(COUNT);}

cover           { return(COVER);}

delete          { return(DELETE);}

display         { return(DISPLAY);}

each            { return(EACH);}

else            { return(ELSE);}

end             { return(END_Q);}

end_if          { return(END_IF);}

end_loop        { return(END_LOOP);}

find_many       { return(FIND_MANY);}

find_one        { return(FIND_ONE);}

float           { sym_flag = 2 ; return(FLOAT);}

for             { return(FOR);}

if              { return(IF);}

in              { return(IN);}

inherit         { return(INHERIT);}
```

```
insert          { return(INSERT);}

integer         { sym_flag = 3; return(INTEGER);}

inverse_of      { return(INVERSE_OF);}

is              { return(IS);}

max             { return(MAX);}

min             { return(MIN);}

mod             { return(MOD);}

not             { return(NOT);}

null            { return(NULLL);}

or              { yylval.t_str = strdup(yytext);return(LOGICAL_OPERATOR);}

project         { return(PROJECT);}

read_input      { return(READ_INPUT);}

set_of          { return(SET_OF);}

string          { return(STRING);}

then            { return(THEN);}

query           { return(QUERY);}

where           { return(WHERE);}

obj_ref         { sym_flag = 4; return(OBJ_REF);}

obj_set         { sym_flag = 5; return(OBJ_SET);}

\:=              { return(ASSIGNMENT_OPERATOR);}

"/="            { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

"<="            { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

">="            { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

"="             { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

"<"             { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

">"             { yylval.t_str = strdup(yytext);return(RELATION_OPERATOR);}

\[              { return(OPEN_BRACKET);}
```

```
\]             { return(CLOSE_BRACKET);}

\{             { return(OPEN_BRACE);}

\}             { return(CLOSE_BRACE);}

\(             { return(OPEN_PARENTHESIS);}

\)             { return(CLOSE_PARENTHESIS);}

\;             { sym_flag = 0;

                 return(SEMICOLON);}

\,             { return(COMMA);}

\:             { return(COLON);}

[*/]           { return(MULTIPLICATION_OPERATOR);}

[+-]           { return(ADDITION_OPERATOR);}

\'[^\']*\'     { yylval.t_str = strdup(yytext);

                 return (STRING_CONSTANT);}


[-\+]?[0-9]+[0-9]*    { yylval.t_str = strdup(yytext);

                        return (INTEGER_CONSTANT);}


[-\+]?[0-9]+\.?[0-9]*    {yylval.t_str= strdup(yytext);

                          return (FLOAT_CONSTANT);}


[A-Za-z][A-Za-z0-9]*([_][A-Za-z0-9]+)*(\.[A-Za-z][A-Za-z0-9]*([_][A-Za-z0-

9]+)*)*(\.[A-Za-z][A-Za-z0-9]*([_][A-Za-z0-9]+)*)? {

                    /* checks to see if "symflag" has a value other than 0 */

                          if (sym_flag) {

                   /* calls the function "symtablook()" */

                              yylval.symp = symtablook(yytext);

                             }
```

```
                                yylval.t_str = strdup(yytext);

                                return(ID);}


\n              yylineno++;

.               printf("invalid character or token encountered at:  %s\n",

yytext);

%%


/*****************************************************************************

function name:symtablook()

return type: struct symtabl*

purpose: Constructs the symbol table by first checking to see if a variable

         has already been entered into the symbol table. If not then id adds

         it to the symbol table.

*****************************************************************************/


struct symtabl*

symtablook(s)

char *s;

{

   struct symtabl *sptr;


   for (sptr = symtabl; sptr < &symtabl[SIZE]; sptr++) {

        /* Checks to see if it is already there. */

       if (sptr->name && !strcmp(sptr->name, s))

           return sptr;
        /* Checks to see if it is free */
        if (!sptr->name) {
            sptr->name = strdup(s);
            switch (sym_flag)
                  {  case (1) : { sptr->type = strdup("char_string");
                                  break;}
                     case (2) : { sptr->type = strdup("float");
                                  break;}
                     case (3) : { sptr->type = strdup("integer");
                                  break;}
                     case (4) : { sptr->type = strdup("objr");
                                  break;}
                 case (5) : { sptr->type = strdup("objs");
                                  break;}
            }
            return sptr;
        }
        /* otherwise continue to next */
    }
    yyerror (" Too many symbols\n");
    exit(1);  /* cannot continue */
} /* symtablook */
```

APPENDIX C - DML YACC FILE

```
/***************************************************************************
file name: dml_yacc.y

purpose: This file contains the semantic definitions of the DML grammer which

         are the input to the YACC Unix tool and the function call to

         construct the intermediate language table.

***************************************************************************/


%{

#include <stdio.h>

#include <string.h>

#include "dml_comp.h"

#include "intmed_tabl.h"


#define   SIMPLE_ST   1  /* Simple statement */

#define   STRUCT_ST   0  /* Structured statement */

#define   INSERT_ST   3  /* Insert statement */

#define   DELETE_ST   3  /* Delete statement */

#define   FIND_ST     4  /* Find_one or Find_many statement */

#define   INPUT_ST    4  /* Read_input statement */

#define   ADDSMP_ST   5  /* Add statement as simple statement */

#define   OUTPUT_ST   5  /* Display or Project statement */

#define   EXPR_ST     6  /* Expression statement */

#define   ADDSTR_ST   6  /* Add statement as structured statement */

#define   NULL        0
int i = 0;

int j = 0;

int k = 0;

int z;

char* s[80];  /* Array for storing simple statements */

char* st[80];  /* Array for storing structured statements */

char text[80];  /* String for storing text */

int N, m, t;

int type;  /* Further defines a type_assign or type_struct statement */

type = SIMPLE_ST;

int type_assign;  /* Stores the type of simple statement */

type_assign = NULL;

int type_struct;  /* Stores the type of structured statement */

type_struct = NULL;

%}


%union {

struct symtabl *symp;

char *t_str;

int t_int;

}




%token <t_int> ADD

%token <t_int> AVG

%token <t_int> BEGIN_Q
```

```
%token <t_int> CHAR

%token <t_int> CHAR_STRING

%token <t_int> CLASS

%token <t_int> CONTAINS

%token <t_int> COUNT

%token <t_int> COVER

%token <t_int> DELETE

%token <t_int> DISPLAY

%token <t_int> EACH

%token <t_int> ELSE

%token <t_int> END_Q

%token <t_int> END_IF

%token <t_int> END_LOOP

%token <t_int> FIND_MANY

%token <t_int> FIND_ONE

%token <t_int> FLOAT

%token <t_int> FOR

%token <t_int> IF

%token <t_int> IN

%token <t_int> INHERIT

%token <t_int> INSERT

%token <t_int> INTEGER

%token <t_int> INVERSE_OF

%token <t_int> IS

%token <t_str> LOGICAL_OPERATOR

%token <t_int> MAX

%token <t_int> MIN
```

```
%token <t_int> MOD

%token <t_int> NOT

%token <t_int> NULLL

%token <t_int> PROJECT

%token <t_int> READ_INPUT

%token <t_int> SET_OF

%token <t_int> STRING

%token <t_int> THEN

%token <t_int> QUERY

%token <t_int> WHERE

%token <t_int> OBJ_REF

%token <t_int> OBJ_SET

%token <t_int> ASSIGNMENT_OPERATOR

%token <t_str> RELATION_OPERATOR

%token <t_int> OPEN_BRACKET

%token <t_int> CLOSE_BRACKET

%token <t_int> OPEN_BRACE

%token <t_int> CLOSE_BRACE

%token <t_int> OPEN_PARENTHESIS

%token <t_int> CLOSE_PARENTHESIS

%token <t_int> SEMICOLON

%token <t_int> COMMA

%token <t_int> COLON

%token <t_int> MULTIPLICATION_OPERATOR

%token <t_int> ADDITION_OPERATOR

%token <t_str> STRING_CONSTANT

%token <t_str> INTEGER_CONSTANT
```

```
%token <t_str> FLOAT_CONSTANT

%token <t_str> ID



%start start1


%%


start1                  : QUERY ID IS declarative_part body_part;



declarative_part        :

                        | declaration declarative_part ;


declaration             : variable_decl ;


variable_decl           : type id_list SEMICOLON ;


id_list                 :  ID {if (type_struct == INPUT_ST || type_struct ==

OUTPUT_ST)

                              st[j++]=strdup($1);} id_lists ;


id_lists                :

                        | COMMA {if (type_struct == INPUT_ST || type_struct

== OUTPUT_ST)

                                st[j++]=strdup("comma");} id_list ;
```

```
type                    : OBJ_REF

                        | OBJ_SET

                        | INTEGER

                        | FLOAT

                        | CHAR_STRING ;


body_part               : BEGIN_Q statement_list END_Q SEMICOLON

{display_intmed();query_constructor();};


statement_list          : { type = SIMPLE_ST;}statement statement_lists ;


statement_lists         :

                        | statement_list ;


statement               : simple_statement SEMICOLON {

                                    switch (type_assign) {

                                        case (INSERT_ST) :{

call_insert(s, i);

                                                            break;}

                                        case (FIND_ST)   :{

call_find(s, i);

                                                            break;}

                                        case (ADDSMP_ST) :{ call_add(s,

i);

                                                            break;}

                                        case (EXPR_ST)   :{

call_expression(s, i);
```

```
                                                            break;}
                                                              }
                                          type_assign = NULL;
                                          for(z=0; z< i;z++){
                                              s[z]= NULL;}
                                          i= 0;};


                         | {type=STRUCT_ST;}structured_statement SEMICOLON {
                                         switch (type_struct) {


                                            case (DELETE_ST):{
call_delete(st, j);
                                                             break;}
                                            case (INPUT_ST) :{
call_input(st, j);
                                                             break;}
                                            case (OUTPUT_ST):{
call_output(st, j);
                                                             break;};
                                           case (ADDSTR_ST):{ call_add(st,
j);
                                                             break;}
                                              }
                                          type_struct = NULL;
                                          type=SIMPLE_ST;
                                          for(z=0; z< j;z++){
                                              st[z]= NULL;}
```

```
                                                j= 0;};



simple_statement        : ID {s[i++]=strdup($1);}simple_statements

                        | NULLL ;


simple_statements       : actual_parameters

                        | ASSIGNMENT_OPERATOR

{s[i++]=strdup("assign");}assign_statement  ;


actual_parameters       :

                        | OPEN_PARENTHESIS actual_parameter_list

CLOSE_PARENTHESIS ;


actual_parameter_list   : expression actual_parameter_lists ;


actual_parameter_lists :

                        | COMMA actual_parameter_list ;

                        ,
structured_statement    : if_statement

                        | loop_statement

                        | delete_statement

                        | input_statement

                        | output_statement

                        | add_statement;


assign_statement        : insert_statement
```

```
                        | find_statement

                        | add_statement

                        | expression ;


if_statement            : IF{st[j++]=strdup("if");} expression

THEN{st[j++]=strdup("then");

                                                        call_if

(st, j);j= 0;}


                          statement_list else_part END_IF

{st[j++]=strdup("end_if");

                                                 call_if(st,

j);j=0;type=1;};


else_part               :

                        | ELSE{st[j++]=strdup("else");call_if(st, j); j=0;}

statement_list;


loop_statement          : FOR EACH ID IN ID {st[k++]=strdup("jump");

st[k++]=strdup($3);

                                       st[k++]=strdup("in");

st[k++]=strdup($5);

                                       call_loop(st, k); k=0;}


                          statement_list END_LOOP {st[k++]=strdup("end_loop");

                                            call_loop(st, k);k=0;};
```

```
insert_statement        : INSERT ID {type_assign =
INSERT_ST;s[i++]=strdup("insert");
                                       s[i++]=strdup($2);} ;


delete_statement        : DELETE OPEN_PARENTHESIS {type_struct=DELETE_ST;
                                                   st[j++]=strdup("delete");
                                                   st[j++]=strdup("opar");}


                          delete_parameter_list
CLOSE_PARENTHESIS{st[j++]=strdup("cpar");};


delete_parameter_list   : ID {st[j++]=strdup($1);}
                        | ID COMMA ID
{st[j++]=strdup($1);st[j++]=strdup("comma");
                                         st[j++]=strdup($3);};


find_statement          : FIND_ONE ID WHERE {type_assign = FIND_ST;
s[i++]=strdup("find_one");
                                              s[i++]=strdup($2);
                                              s[i++]=strdup("where");}
expression


                        | FIND_MANY ID {type_assign = FIND_ST;
s[i++]=strdup("find_many");
                                          s[i++]=strdup($2);} where_expr ;


where_expr              :
```

```
                                 |WHERE {s[i++]=strdup("where");} expression ;



input_statement         : READ_INPUT OPEN_PARENTHESIS{type_struct=INPUT_ST;

st[j++]=strdup("read_input");
                                                          st[j++]=strdup("opar");}

                                   id_list CLOSE_PARENTHESIS { st[j++]=strdup("cpar");};

output_statement        : PROJECT ID {type_struct=5;st[j++]=strdup("project");
                                            st[j++]=strdup($2);}

                        | DISPLAY OPEN_PARENTHESIS {type_struct=OUTPUT_ST;
                                                        st[j++]=strdup("display");
                                                        st[j++]=strdup("opar");}

                          display_parameter_list
CLOSE_PARENTHESIS{st[j++]=strdup("cpar");};

display_parameter_list : id_list
                        | literal ;

add_statement           : ADD OPEN_PARENTHESIS ID COMMA ID CLOSE_PARENTHESIS
{if (type){
                          s[i++]=strdup("add");
s[i++]=strdup("opar");s[i++]=strdup($3);
s[i++]=strdup("comma");s[i++]=strdup($5);s[i++]=strdup("cpar");

                         type_assign = ADDSMP_ST;}


                         else {

                           st[j++]=strdup("add"); st[j++]=strdup("opar");

                           st[j++]=strdup($3);st[j++]=strdup("comma");

                           st[j++]=strdup($5);st[j++]=strdup("cpar");

                           type_struct = ADDSTR_ST;}};


statistical_statement  : MIN OPEN_PARENTHESIS ID CLOSE_PARENTHESIS {if

(type) {

                         s[i++]=strdup("min");s[i++]=strdup("opar");

                         s[i++]=strdup($3);s[i++]=strdup("cpar");}


                         else {

                           st[j++]=strdup("min");st[j++]=strdup("opar");

                           st[j++]=strdup($3);st[j++]=strdup("cpar");}}


                       | MAX OPEN_PARENTHESIS ID CLOSE_PARENTHESIS {if

(type) {

                         s[i++]=strdup("max");s[i++]=strdup("opar");

                         s[i++]=strdup($3);s[i++]=strdup("cpar");}


                         else {

                           st[j++]=strdup("max");st[j++]=strdup("opar");

                           st[j++]=strdup($3);st[j++]=strdup("cpar");}}
                         | AVG OPEN_PARENTHESIS ID CLOSE_PARENTHESIS {if
(type) {
                           s[i++]=strdup("avg");s[i++]=strdup("opar");
                           s[i++]=strdup($3);s[i++]=strdup("cpar");}

                           else {
                             st[j++]=strdup("avg");st[j++]=strdup("opar");
                             st[j++]=strdup($3);st[j++]=strdup("cpar");}}

                         | COUNT OPEN_PARENTHESIS ID  CLOSE_PARENTHESIS {if
(type) {
                           s[i++]=strdup("count");s[i++]=strdup("opar");
                           s[i++]=strdup($3);s[i++]=strdup("cpar");}

                           else {
                             st[j++]=strdup("count");st[j++]=strdup("opar");
                             st[j++]=strdup($3);st[j++]=strdup("cpar");}};
                        ,
expression               : {if ((!type_assign)&& (type)) {
                               type_assign = EXPR_ST;}} rel_expr expressions ;

expressions              :
                         | LOGICAL_OPERATOR { if (type) {
                                                 s[i++] =strdup($1);}

                                             else {
                                                  st[j++] =strdup($1);}}
expression
                              | contain_expr ;

rel_expr                      : simple_expr rel_exprs
                              | statistical_statement rel_exprs;

rel_exprs                     :
                              | RELATION_OPERATOR {if (type) {
                                                    s[i++]=strdup($1);}

                                               else {
                                                  st[j++]=strdup($1);}}
simple_expr ;

contain_expr                  : CONTAINS { if (type){
                                             s[i++] =strdup("contains");}

                                         else {
                                           st[j++]=strdup("contains");}}
contain_exprs;


contain_exprs                 : expression;

simple_expr                   : term simple_exprs ;
```

```
simple_exprs            :
                        | ADDITION_OPERATOR {if (type){
                                                  s[i++]=strdup("addop");}


                                              else {
                                                st[j++]=strdup("addop");}}
simple_expr ;


term                    : factor terms ;


terms                   :
                        | MULTIPLICATION_OPERATOR {if (type){
                                                        s[i++]=strdup("multop");}


                                                    else {


st[j++]=strdup("multop");}} term ;


factor                  : NOT {if (type){
                                  s[i++]= strdup("not");}


                               else {
                                 st[j++]=strdup("not");}} primary
                        | primary ;


primary                 : literal
                        | ID {if (type){
                                s[i++]=strdup($1);}


                             else {

                             st[j++]=strdup($1);}}


                         | OPEN_PARENTHESIS {if (type){

                                              s[i++]=strdup("opar");}


                                           else {

                                             st[j++]=strdup("opar");}}

expression CLOSE_PARENTHESIS

                           {if (type){

                               s[i++] = strdup("cpar");}


                            else {

                              st[j++]= strdup("cpar");}};


literal                  : INTEGER_CONSTANT {if (type){

                                              s[i++]= strdup($1);}


                                           else {

                                             st[j++]= strdup($1);}}


                         | FLOAT_CONSTANT  {if (!type){

                                             s[i++]= strdup($1);}


                                          else {
```

```
                                      st[j++]= strdup($1);}}
                        | STRING_CONSTANT {strcpy(text, $1);
                                      N = strlen(text);
                                      t = 0;
            /* This extracts single quotations from text. */
                                      for(m = 1; m< (N-1); m++)
                                         text[t++] = text[m];
                                      text[t]= '\0';
                                      if (type){
                                        s[i++]= strdup(text);}


                                      else {
                                        st[j++]= strdup(text);}};




%%
#include <stdio.h>



extern int dmllineno;


void dmlerror(s)
char* s;


{
  fflush(stdout);
  fflush(stderr);
  fprintf(stderr, "%s at line %d\n",s,dmllineno);
}


}
```

APPENDIX D- RULES TO FILL IN THE INTERMEDIATE TABLE

The following pages illustrate the correct procedure for filling in the intermediate language table. The first line of each example list the Object-oriented operation and its equivalent funtion name in the file “intmed_tabl.c” which contains the actual code to fill in the intermediate language table. The next line of each example lists a generic instance of the Object-oriented operation. That operation is then followed by a conceptual illustration of the intermediate language table. The information in the table shows how the pertinent information from the operation is to be distributed in the columns and rows of the table.

Find_one : [See function “call_find()” in file “intmed_tabl.c”]

id_a := Find_one id_b where expression

Operation	$Argument_1$	$Argument_2$	Result
Find_one	id_b		id_a
(expression)			
relational_operator	term	term	t_1
relational_operator	term	term	t_2
logical_operator	t_1	t_2	0

Find_many : [See function "call_find()" in file "intmed_tabl.c"]

id_a := Find_many id_b where expression

Operation	$Argument_1$	$Argument_2$	Result
Find_many	id_b		id_a

Contains : [See function "call_find()" in file "intmed_tabl.c"]

id_a contains id_b

Operation	$Argument_1$	$Argument_2$	Result
contains	id_a	id_b	

id_a contains rel_exp

Operation	$Argument_1$	$Argument_2$	Result
contains	id_a		
relational_operator	term	term	t_1
relational_operator	term	term	t_2
logical_operator	t_1	t_2	0

id_a contains literal

Operation	$Argument_1$	$Argument_2$	Result
contains	id_a	literal	

Assignment : [See function "call_expression()" in file "intmed_tabl.c"]

$id_a := id_b$

Operation	Argument_1	Argument_2	Result
assign	id_b		id_a

Insert : [See function "call_find()" in file "intmed_tabl.c"]

id_a := Insert id_b

Operation	Argument_1	Argument_2	Result
insert	id_b		id_a

Delete : [See function "call_find()" in file "intmed_tabl.c"]

delete id_a

Operation	Argument_1	Argument_2	Result
delete	id_a		

delete (id_a, id_b)

Operation	Argument_1	Argument_2	Result
delete	id_a	id_b	

Read_input : [See function “call_find()” in file “intmed_tabl.c”]

read_input   id_list

Operation	Argument$_1$	Argument$_2$	Result
read_input	(id_list)		

(id_list)

	id$_a$	id$_b$	1
	id$_c$	id$_d$	0

Add : [See function “call_find()” in file “intmed_tabl.c”]

add (id$_a$, id$_b$)

Operation	Argument$_1$	Argument$_2$	Result
add	id$_a$	id$_b$	

Display   (attribute) : [See function “call_find()” in file “intmed_tabl.c”]

display (id)

Operation	Argument$_1$	Argument$_2$	Result
display	id		

display (string constant)

Operation	Argument_1	Argument_2	Result
display	string_constant		

Project : [See function “call_find()” in file “intmed_tabl.c”]

project id

Operation	Argument_1	Argument_2	Result
project	id		

Loop_statement: [See function “call_find()” in file “intmed_tabl.c”]

For Each id IN id statement_list End_Loop

Operation	**Argument_1**	**Argument_2**	**Result**
Jump	Id	Id(set of Id)	F1 *push on the stack
Statement-list			
End_Loop			

If: [See function "call_find()" in file "intmed_tabl.c"]

If expression then statement_list else_part End_if
else_part : E | ELSE statement_list

Operation	Argument_1	Argument_2	Result
If			C1

(expression)

relational_operator	term	term	t1
relational_operator	term	term	t2
logical_operator	t1	t2	

(statement_list)

else_part : E | else_part

Operation	Argument_1	Argument_2	Result
else			C1

(statement_list)

end_if			

APPENDIX E - INTERMEDIATE TABLE HEADER FILE

```
/**************************************************************************

file name: intmed_tabl.h

purpose: This file contains the declarations for "intmed_tabl.c" and the

         definition of the "intmed_node" structure.

**************************************************************************/


struct intmed_node {

       int    lineno;

       char   *opar;

       char   *arg1;

       char   *arg2;

       char   *result;

       struct intmed_node  *next;

       struct intmed_node  *prior;

     };


void call_insert();


void call_find();


void call_delete();


void call_expression();


void call_if();
void call_loop();


void call_input();


void call_output();


void call_add();


struct intmed_node  *intmed_node_alloc();


void display_intmed ();
```

APPENDIX F - INTERMEDIATE TABLE IMPLEMENTATION FILE

```
/*****************************************************************************

file name: intmed_tabl.c

purpose: This file contains the structures and functions to construct the

         Intermediate language table.

*****************************************************************************/


#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include "intmed_tabl.h"



struct intmed_node  *intmed_first_node;

struct intmed_node  *temptr;

struct intmed_node  *priptr;  /* Points to the previous node. */


struct intmed_node  *temp1_ptr;

struct intmed_node  *temp2_ptr;


char* temp_arr[9];  /* String array for temporary storage of results in the

                       intermediate language table. */



int q = 0;
int l = 0;



/****************************************************************************
function name: call_insert()
return type: void
purpose: This function fills in the table for the OO insert operation.
****************************************************************************/
void call_insert(arr_ins, size)
char** arr_ins;
int size;
{

  /* This code can be used to understand the coding structure for
call_insert()
  int m;
  printf("\nStructure for insert_statement\n");
  for(m = 0; m < size; m++)
     printf("%d %s\n", m, arr_ins[m]);*/

/* Temporary variables for the storage of temporary results which are limited
   to a total of 9.  Not limited by any other factor. */
  temptr = intmed_node_alloc();
  temptr -> lineno = q++;
  temptr -> opar = arr_ins[2];
  temptr -> arg1 = arr_ins[3];
  temptr -> result = arr_ins[0];
  temptr -> prior = priptr;

  if (priptr)

        priptr -> next = temptr;

  priptr = temptr;


}/* end of call_insert() */


/**************************************************************************

function name: call_find()

return type: void

purpose: This function fills in the table for both OO find_one and find_many

operations.  NOTE: Limiting factor is max size for find statement is as

follows:  a := find_one/find_many where (a rel_expr b) logical_expr

(c rel_expr d) logical_expr (e rel_expr f)


**************************************************************************/


void call_find(arr_fin, size)

char** arr_fin;

int size;

{


  int j = 0;



  /* This code can be used to understand the coding structure for call_find()

   int m;
```

```
printf("\nStructure for find_statement\n");
for(m = 0; m < size; m++)
   printf("%d %s\n", m, arr_fin[m]);*/




temp_arr[0] = "t0";
temp_arr[1] = "t1";
temp_arr[2] = "t2";
temp_arr[3] = "t3";
temp_arr[4] = "t4";
temp_arr[5] = "t5";
temp_arr[6] = "t6";
temp_arr[7] = "t7";
temp_arr[8] = "t8";

temptr = intmed_node_alloc();
temptr -> lineno = q++;
temptr -> opar = arr_fin[2];
temptr -> arg1 = arr_fin[3];
temptr -> result = arr_fin[0];
temptr -> prior = priptr;
if (priptr)
   priptr -> next = temptr;
priptr = temptr;
```

```
if ( size == 8 ) {

     temptr = intmed_node_alloc();

     temptr -> lineno = q++;

     temptr -> opar = arr_fin[6];

     temptr -> arg1 = arr_fin[5];

     temptr -> arg2 = arr_fin[7];

     temptr -> result = "0";

     temptr -> prior = priptr;

     priptr -> next = temptr;

     priptr = temptr;

     l = 0;

   }


j = 8;

if ( (size == 12) || (size == 16)) {

      temptr = intmed_node_alloc();

      temptr -> lineno = q++;

      temptr -> opar = arr_fin[j - 2];

      temptr -> arg1 = arr_fin[j - 3];

      temptr -> arg2 = arr_fin[j - 1];

      temptr -> result = temp_arr[l++];

      temptr -> prior = priptr;

      priptr -> next = temptr;

      priptr = temptr;


      temptr = intmed_node_alloc();

      temptr -> lineno = q++;
```

```
 temptr -> opar = arr_fin[j + 2];

 temptr -> arg1 = arr_fin[j + 1];

 temptr -> arg2 = arr_fin[j + 3];

 temptr -> result = temp_arr[l++];

 temptr -> prior = priptr;

 priptr -> next = temptr;

 temp1_ptr = priptr;

 priptr = temptr;


 temptr = intmed_node_alloc();

 temptr -> lineno = q++;

 temptr -> opar = arr_fin[j];

 temptr -> arg1 = temp1_ptr -> result;

 temptr -> arg2 = priptr -> result;

 if (size == 16)

     temptr -> result = temp_arr[l++];

 else {

      temptr -> result = "0";

}

 temptr -> prior = priptr;

 priptr -> next = temptr;

 priptr = temptr;

 temp2_ptr = temptr;

}

if (size == 16 ) {

    temptr = intmed_node_alloc();

    temptr -> lineno = q++;
    temptr -> opar = arr_fin[j + 6];

    temptr -> arg1 = arr_fin[j + 5];

    temptr -> arg2 = arr_fin[j + 7];

    temptr -> result = temp_arr[l++];

    temptr -> prior = priptr;

    priptr -> next = temptr;

    temp1_ptr = priptr;

    priptr = temptr;


    temptr = intmed_node_alloc();

    temptr -> lineno = q++;

    temptr -> opar = arr_fin[j + 4];

    temptr -> arg1 = temp2_ptr -> result;

    temptr -> arg2 = priptr -> result;

    temptr -> result = "0";

    temptr -> prior = priptr;

    priptr -> next = temptr;

    priptr = temptr;

 }


if ( size == 20 ) {

     temptr = intmed_node_alloc();

     temptr -> lineno = q++;

     temptr -> opar = arr_fin[j - 2];

     temptr -> arg1 = arr_fin[j - 3];

     temptr -> arg2 = arr_fin[j - 1];

     temptr -> result = temp_arr[l++];
temptr -> prior = priptr;

priptr -> next = temptr;

priptr = temptr;


temptr = intmed_node_alloc();

temptr -> lineno = q++;

temptr -> opar = arr_fin[j + 2];

temptr -> arg1 = arr_fin[j + 1];

temptr -> arg2 = arr_fin[j + 3];

temptr -> result = temp_arr[l++];

temptr -> prior = priptr;

priptr -> next = temptr;

temp1_ptr = priptr;

priptr = temptr;


temptr = intmed_node_alloc();

temptr -> lineno = q++;

temptr -> opar = arr_fin[j];

temptr -> arg1 = temp1_ptr -> result;

temptr -> arg2 = priptr -> result;

temptr -> result = temp_arr[l++];

temptr -> prior = priptr;

priptr -> next = temptr;

priptr = temptr;

temp2_ptr = temptr;


temptr = intmed_node_alloc();
temptr -> lineno = q++;

temptr -> opar = arr_fin[j + 6];

temptr -> arg1 = arr_fin[j + 5];

temptr -> arg2 = arr_fin[j + 7];

temptr -> result = temp_arr[l++];

temptr -> prior = priptr;

priptr -> next = temptr;

temp1_ptr = priptr;

priptr = temptr;


temptr = intmed_node_alloc();

temptr -> lineno = q++;

temptr -> opar = arr_fin[j + 10];

temptr -> arg1 = arr_fin[j + 9];

temptr -> arg2 = arr_fin[j + 11];

temptr -> result = temp_arr[l++];

temptr -> prior = priptr;

priptr -> next = temptr;

temp1_ptr = priptr;

priptr = temptr;


temptr = intmed_node_alloc();

temptr -> lineno = q++;

temptr -> opar = arr_fin[j + 8];

temptr -> arg1 = temp1_ptr -> result;

temptr -> arg2 = priptr -> result;

temptr -> result = temp_arr[l++];
```

```
            temptr -> prior = priptr;

            priptr -> next = temptr;

            priptr = temptr;


            temptr = intmed_node_alloc();

            temptr -> lineno = q++;

            temptr -> opar = arr_fin[j + 4];

            temptr -> arg1 = temp2_ptr -> result;

            temptr -> arg2 = priptr -> result;

            temptr -> result = "0";

            temptr -> prior = priptr;

            priptr -> next = temptr;

            priptr = temptr;

        }



   l = 0;


} /* end of call_find() */


/*****************************************************************************

function name: call_delete()

return type: void

purpose: This function fills in the table for the OO delete operations of one

and two arguments.

*****************************************************************************/
```

```
void call_delete(arr_del, size)

char** arr_del;

int size;

{


 /* This code can be used to understand the coding structure for call_delete()

  int m;

  printf("\nStructure for delete_statement\n");

  for(m = 0; m < size; m++)

     printf("%d %s\n", m, arr_del[m]);*/



  temptr = intmed_node_alloc();

  temptr -> lineno = q++;

  temptr -> opar = arr_del[0];

  temptr -> arg1 = arr_del[2];

  if ( arr_del[4] )

       temptr -> arg2 = arr_del[4];

  temptr -> prior = priptr;

  if (priptr)

      priptr -> next = temptr;

  priptr = temptr;


 } /* end of call_delete() */
```

```
/***********************************************************************

function name: call_expression()

return type: void

purpose: This function fills in the table on assignments of values to

variables.

***********************************************************************/


void call_expression(arr_exp, size)

char** arr_exp;

int size;

{


 /*This code can be used to understand the coding structure for

call_expression()

  int m;

  printf("\nStructure for expression\n");

  for(m = 0; m < size; m++)

     printf("%d %s\n", m, arr_exp[m]);*/


  temptr = intmed_node_alloc();

  temptr -> lineno = q++;

  temptr -> opar = arr_exp[1];

  temptr -> arg1 = arr_exp[2];

  temptr -> result = arr_exp[0];

  temptr -> prior = priptr;

  if (priptr)
```

```
        priptr -> next = temptr;
    priptr = temptr;

} /* end of call_expression */

/*****************************************************************************
function name: call_if()
return type: void
purpose: This function fills in the table for the OO if operations. NOTE:
Limitation of two relative expressions per each if statement.  No limiting
factors.
*****************************************************************************/

void call_if(arr_if, size)
char** arr_if;
int size;
{

 /*This code can be used to understand the coding structure for call_if()
  int m;
  printf("\nStructure for if_statement\n");
  for(m = 0; m < size; m++)
     printf("%d %s\n", m, arr_if[m]);*/


  temptr = intmed_node_alloc();
  temptr -> lineno = q++;
temptr -> opar = arr_if[0];
temptr -> prior = priptr;
if (priptr)
    priptr -> next = temptr;
priptr = temptr;
if ( size == 4 ) {
     temptr = intmed_node_alloc();
     temptr -> lineno = q++;
     temptr -> opar = arr_if[2];
     temptr -> arg1 = arr_if[1];
     temptr -> arg2 = arr_if[3];
     temptr -> result = temp_arr[l++];
     temptr -> prior = priptr;
     if (priptr)
         priptr -> next = temptr;
     priptr = temptr;
     l = 0;
   }
if ( (size == 8) && (!strcmp(arr_if[2], "opar"))) {
     temptr = intmed_node_alloc();
     temptr -> lineno = q++;
     temptr -> opar = arr_if[1];
     temptr -> arg1 = arr_if[3];
     temptr -> result = temp_arr[l++];
     temptr -> prior = priptr;
     priptr -> next = temptr;
     priptr = temptr;
```

```
        temptr = intmed_node_alloc();

        temptr -> lineno = q++;

        temptr -> opar = arr_if[5];

        temptr -> arg1 = priptr -> result;

        temptr -> arg2 = arr_if[6];

        temptr -> result = temp_arr[l++];

        temptr -> prior = priptr;

        priptr -> next = temptr;

        priptr = temptr;

        l = 0;

    }

 else {

        if ( (size == 9) && (!strcmp(arr_if[4], "and") || !strcmp(arr_if[4],

"or" ))) {

             temptr = intmed_node_alloc();

             temptr -> lineno = q++;

             temptr -> opar = arr_if[2];

             temptr -> arg1 = arr_if[1];

             temptr -> arg2 = arr_if[3];

             temptr -> result = temp_arr[l++];

             temptr -> prior = priptr;

             priptr -> next = temptr;

             priptr = temptr;


             temptr = intmed_node_alloc();

             temptr -> lineno = q++;
```

```
            temptr -> opar = arr_if[6];

            temptr -> arg1 = arr_if[5];

            temptr -> arg2 = arr_if[7];

            temptr -> result = temp_arr[l++];

            temptr -> prior = priptr;

            priptr -> next = temptr;

            temp1_ptr = priptr;

            priptr = temptr;


            temptr = intmed_node_alloc();

            temptr -> lineno = q++;

            temptr -> opar = arr_if[4];

            temptr -> arg1 = temp1_ptr -> result;

            temptr -> arg2 = priptr -> result;

            temptr -> result = temp_arr[l++];

            temptr -> prior = priptr;

            priptr -> next = temptr;

            priptr = temptr;

            l = 0;

         }


     else {

           }

   }


} /* end of call_if() */
```

```
/*****************************************************************************

function name: call_loop()

return type: void

purpose: This function fills in the table for the OO loop operations.

*****************************************************************************/


void call_loop(arr_loop, size)

char** arr_loop;

int size;

{


 /* This code can be used to understand the coding structure for call_loop()

  int m;

  printf("\nStructure for loop_statement\n");

  for(m = 0; m < size; m++)

     printf("%d %s\n", m, arr_loop[m]);*/


  temptr = intmed_node_alloc();

  temptr -> lineno = q++;

  temptr -> opar = arr_loop[0];

  if ( size > 1 ) {

        temptr -> arg1 = arr_loop[1];

        temptr -> arg2 = arr_loop[3];

     }

  temptr -> prior = priptr;

  if (priptr)
```

```
      priptr -> next = temptr;
  priptr = temptr;

} /* end of call_loop() */

/*****************************************************************************
function name: call_input()
return type: void
purpose: This function fills in the table for the OO input operation.
*****************************************************************************/

void call_input(arr_inp, size)
char** arr_inp;
int size;
{
  int m = 0;

 /* This code can be used to understand the coding structure for call_input()
  int p;
  printf("\nStructure for input_statement\n");
  for(p = 0; p < size; p++)
     printf("%d %s\n", p, arr_inp[p]);*/

  for( m = 0; m <= (size - 4); m+=4) {
       temptr = intmed_node_alloc();
       temptr -> lineno = q++;
       temptr -> opar = arr_inp[0];
```

```
        temptr -> arg1 = arr_inp[m + 2];
        if ( (m + 4) <= size )
            temptr -> arg2 = arr_inp[m + 4];
        temptr -> result = "1";
        temptr -> prior = priptr;
        if (priptr)
            priptr -> next = temptr;
        priptr = temptr;
      }
   if (priptr)
       priptr -> result = "0";


} /* end of call_input() */

/******************************************************************************
function name: call_output()
return type: void
purpose: This function fills in the table for the OO output operation


******************************************************************************/

void call_output(arr_out, size)
char** arr_out;
int size;
{
   int m = 0;
```

```
/* This code can be used to understand the coding structure for call_output()
 int p;
 printf("\nStructure for output_statement\n");
 for(p = 0; p < size; p++)
    printf("%d %s\n", p, arr_out[p]);*/


 for( m = 0; m <= (size - 4); m+=4) {
      temptr = intmed_node_alloc();
      temptr -> lineno = q++;
      temptr -> opar = arr_out[0];
      temptr -> arg1 = arr_out[m + 2];
      if ( (m + 4) <= size )
          temptr -> arg2 = arr_out[m + 4];
      temptr -> result = "1";
      temptr -> prior = priptr;
      if (priptr)
          priptr -> next = temptr;
      priptr = temptr;
    }
 if (priptr)
     priptr -> result = "0";


} /* end of call_output() */

/*************************************************************************
function name: call_add()
```

```
return type: void

purpose: This function fills in the table for the OO add operation.

***************************************************************************/


void call_add(arr_add, size)

char** arr_add;

int size;

{


/* This code can be used to understand the coding structure for call_add()

  int p;

  printf("\nStructure for add_statement\n");

  for(p = 0; p < size; p++)

     printf("%d %s\n", p, arr_add[p]);*/


  temptr = intmed_node_alloc();

  temptr -> lineno = q++;

  temptr -> opar = arr_add[0];

  temptr -> arg1 = arr_add[2];

  temptr -> arg2 = arr_add[4];

  temptr -> prior = priptr;

  if (priptr)

      priptr -> next = temptr;

  priptr = temptr;

} /* end of call_add() */
```

```
/*************************************************************************

function name: intmed_node_alloc()

return type: struct intmed_node

purpose: This function allocates an area of memory for the structure of

intmed_node.

*************************************************************************/


struct intmed_node  *intmed_node_alloc()

{

  struct intmed_node  *new_intmed_node;




#ifdef EnExFlag

  printf("Enter intmed_node_alloc\n");

#endif


  if ((new_intmed_node = (struct intmed_node *)

          malloc (sizeof(struct intmed_node))) == 0) {

      printf("*** intmed_node_alloc problem with malloc\n");

      sleep(10);

    }


#ifdef EnExFlag

   printf("Exit intmed_node_alloc\n");

#endif


   new_intmed_node -> lineno = 0;
   new_intmed_node -> opar = "";

   new_intmed_node -> arg1 = "";

   new_intmed_node -> arg2 = "";

   new_intmed_node -> result = "";

   new_intmed_node -> next = 0;

   new_intmed_node -> prior = 0;

   if (!intmed_first_node){

      intmed_first_node = new_intmed_node;

    }

   return (new_intmed_node);

} /* end of intmed_node_alloc()



/*************************************************************************

function name: display_intmed()

return type: void

purpose: This function displays the intermediate language table.


*************************************************************************/


void display_intmed ()

{

  struct intmed_node  *temp;


  temp = intmed_first_node;

  printf("LINENO     OPAR           ARG1          ARG2          RESULT\n");

  while(temp){
    printf("%d\t %s\t %s\t %s\t %s\t\n  ", temp->lineno, temp->opar, temp-
>arg1, temp->arg2, temp->result);
   temp = temp->next;
  }

} /* end of display_intmed() */
```

APPENDIX G - DML QUERY CONSTRUCTOR HEADER FILE

```
/***************************************************************************
file name: dml_comp.h
purpose: This file contains the declarations for the "dml_comp.c" and also
         the definition of the "abdlque_node" structure.
***************************************************************************/
#include "symtabl.h"
#include "intmed_tabl.h"


#define  MAXPATHLEN  10  /* The max length of path expression */
#define  MAXSETVAR   20  /* The max number of compiler-defined objset
variables */
#define  MAXREFVAR   20  /* The max number of compiler-defined objref
variables */
#define  CLSSTKSIZE  10  /* The size of the classname stack */
struct abdlque_node {
       char   text[100];
       struct abdlque_node  *next;
       struct abdlque_node  *prior;
};


struct abdlque_node  *abdlque_node_alloc();


struct dict_ocls_node  *in_dictionary();


struct dict_attr_node  *in_class();
struct symtabl *in_symtab();

struct dict_attr_node  *check_inherit();

struct dict_attr_node  *check_cover();

void query_constructor();

void convert_insert_statement();

void convert_find_statement();

void convert_loop_statement();

void convert_delete_statement();

void convert_assign_statement();

void convert_display_statement();

void convert_if_statement();

void convert_readInput_statement();

void convert_add_statement();
void construct_add();

struct obj_dbid_node *createDict();

char *convert_logical_statement();

char *convert_rel_expr();

char *convert_contains_statement();

void construct_retrieve_gen();

void construct_retrieve_logical();

void construct_retrieve_by();

char *construct_retrieve_ref();

void construct_update();

void construct_retrieve_display();

char *converttoupper();

char *converttoclassname();

char *converttolower();
```

```
void add_varlist();


void display_abdl();


void cleanup_lists();


void input_to_rtm();
```

APPENDIX H - QUERY CONSTRUCTOR IMPLEMENTATION FILE

```
/*****************************************************************************
file name: dml_comp.c
purpose: Interpretation of the intermediate language table, symbol table and
         data dictionary for the construction of ABDL queries which are placed
         in a file for execution by the RTM.
*****************************************************************************/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "licommdata.h"
#include "oodb_structures.h"
#include "dml_comp.h"


extern struct intmed_node  *intmed_first_node; /*defined in intmed_tabl.c */

struct intmed_node  *intmed_ptr;


/*These variables point the the nodes of the ABDL query*/
struct abdlque_node  *abdlque_pri_node;
struct abdlque_node  *abdlque_first_node;
struct abdlque_node  *abdlque_current_node;


/*These variables point to the nodes of the data dictionary*/
struct obj_dbid_node  *obj_dbid_ptr;
struct dict_ocls_node  *dict_ocls_ptr;
struct dict_attr_node  *dict_attr_ptr;

char objref_arr[MAXREFVAR][3];/*used for compiler-defined obj_ref variables*/
char objset_arr[MAXSETVAR][3];/*used for compiler-defined obj_set variables*/
char reftabl_arr[CLSSTKSIZE][ANLength + 1];/*path expression for classnames*/
char attr_arr[MAXPATHLEN][ANLength + 1];/*path expression for attribute
names*/
char equal[2]; /*This variable stores an equality sign*/
char final_result[ANLength + 1];/*This variable stores the variable name of
a find operation*/

int s;  /* s for the index of reftabl_arr */
int a;  /* a for the index of attr_arr */
int f;  /* f for the index of obj_ref_arr */
int p;  /* p for the index of obj_set_arr */
int subsflag;  /* informs retrieve_statement of a substitution */


/*****************************************************************************
funtion name: abdlque_node_alloc()
return type: struct abdlque_node
```

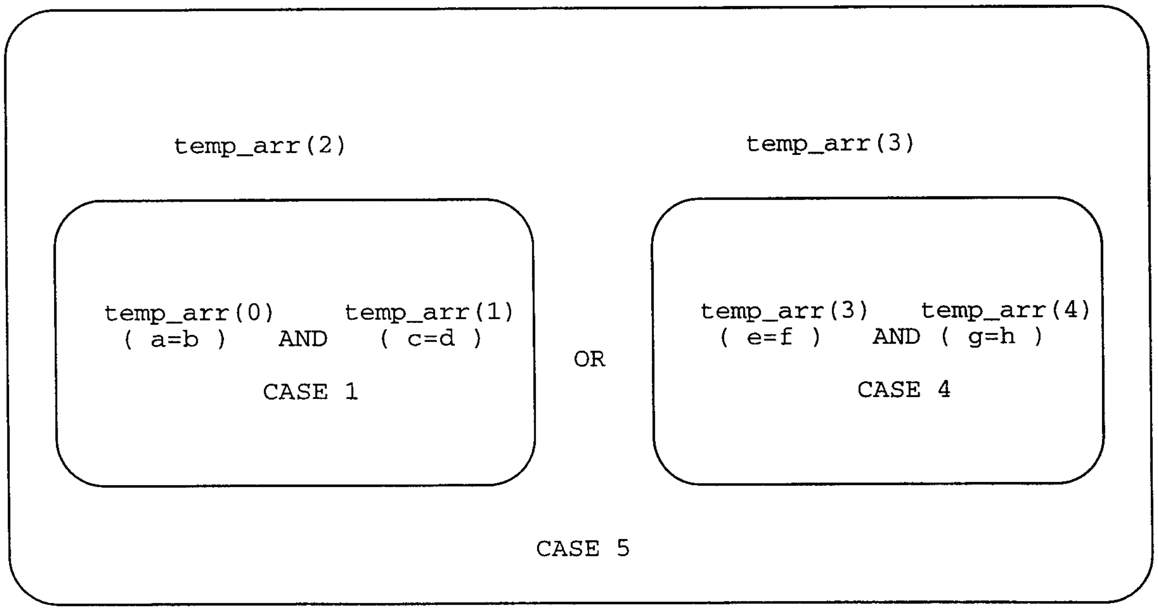
```
purpose: This function allocates memory for the construction of ABDL query
         nodes.
***************************************************************************/
struct abdlque_node  *abdlque_node_alloc()
{
  struct abdlque_node  *new_abdlque_node;

  /* Allocate memory for the structure of an abdlque_node */

  if ((new_abdlque_node = (struct abdlque_node *)
                  malloc (sizeof(struct abdlque_node))) == 0) {
      fprintf (stderr, "***abdlque_node_alloc problem with malloc \n");
      sleep(10);
  }
  strcpy(new_abdlque_node -> text, "");
  new_abdlque_node -> prior = NULL;
  new_abdlque_node -> next = NULL;
  if (!abdlque_first_node){
        abdlque_first_node = new_abdlque_node;
      }
  return(new_abdlque_node);
}/* End of abdlque_node_alloc */

/***************************************************************************
funtion name: query_constructor()
return type: void
purpose: This function serves as the main controller for dml_comp.c
***************************************************************************/
void query_constructor()
{
   int i;
   char z;
   char tempref[3];
   char tempset[3];
   char text[20];

#ifdef EnExFlag
   printf("Enter query_constructor\n");
#endif

   obj_dbid_ptr = createDict(); /*This opens the data dictionary*/
   strcpy(equal, "="); /*it will be used in retrieve statements */
   dict_ocls_ptr = obj_dbid_ptr-> odn_first_dict_cls; /*This points to the
first class of the data dictionary*/
/*This loop initializes the objref_arr for compiler defined obj_ref variable.
 ex: ra,rb */
   z = 'a';
   for(i = 0; i< 20;i++)  {
       tempref[0]= 'r';
       tempref[1]= z++;
       tempref[2]= '\0';
       strcpy(objref_arr[i], tempref);
     }
```

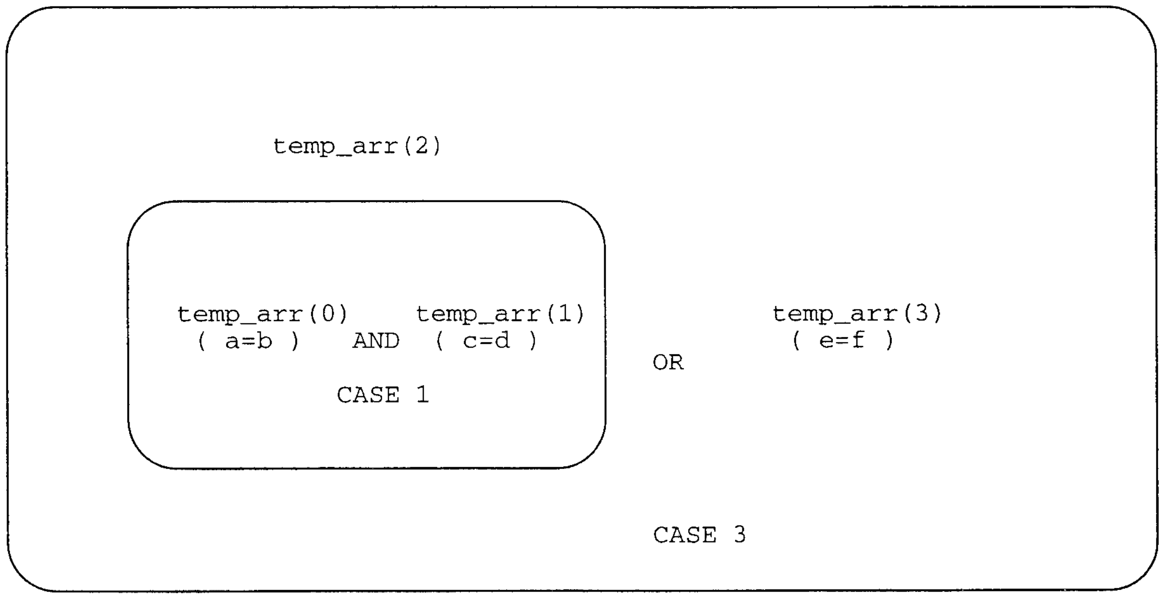
```
/*This loop initializes the objset_arr for compiler defined obj_set variable.
 ex: sa,sb */
   z = 'a';
   for(i = 0; i< 20; i++) {
       tempset[0] = 's';
       tempset[1] = z++;
       tempset[2] = '\0';
       strcpy(objset_arr[i], tempset);
     }

   intmed_ptr = intmed_first_node;  /*Points to the first line of the
intermediate table*/

   s = 0;
   f = 0;
   p = 0;
   while(intmed_ptr) {
        strcpy(final_result, "&"); /* the form needed in abdl-side */
        if(intmed_ptr->result)
          strcat(final_result, intmed_ptr->result); /*last assignment to be
done*/
        strcpy(text, intmed_ptr->opar);
       /*list of possible branches*/
        switch (text[0])
          { case ('f') : {convert_find_statement();
                          break;}

            case ('d') : {if (text[1]== 'i')
                              convert_display_statement();
                          else {
                              convert_delete_statement();
                             }
                          break;}

            case ('a') : {if (text[1] == 'd')
                              convert_add_statement();
                          else {
                              convert_assign_statement();
                            }
                          break;}

            case ('r') : {convert_readInput_statement();
                          break;}

            case ('j') : {convert_loop_statement();
                          break;}

            case ('i') : {if (text[1] == 'n')
                              convert_insert_statement();
                          else{
                              convert_if_statement();
                                    }
                              break;}

                case ('e') : {if (text[4] == 'i')
                                  convert_if_statement();
                              else{
                                  convert_loop_statement();
                               }
                              break;}

            }/* end of switch statement */

        intmed_ptr = intmed_ptr -> next;
        s = 0;
        a = 0;
       }/* end of while statement */
        add_varlist();
        display_abdl();
        input_to_rtm();
        cleanup_lists();

#ifdef EnExFlag
   printf("Exit query_constructor\n");
#endif

}/* end of query_constructor */


/*******************************************************************************
funtion name: in_dictionary(ocname)
return type: struct dict_ocls_node
purpose: This function determines if a classname, given as an arguement, is
         in the data dictionary.  If so, it returns the pointer to that class.
*******************************************************************************/
struct dict_ocls_node *in_dictionary(ocname)
char   ocname[ANLength + 1];
{
  struct dict_ocls_node *dict_ocls_temp;

#ifdef EnExFlag
  printf("Enter in_dictionary\n");
#endif

  dict_ocls_temp = dict_ocls_ptr;

  while(dict_ocls_temp) {
        if( !strcmp(dict_ocls_temp -> dict_ocn_name, ocname) ){
             dict_attr_ptr = dict_ocls_temp -> dict_first_attr;
#ifdef EnExFlag
             printf("Exit in_dictionary true\n");
#endif
             return(dict_ocls_temp);}

        dict_ocls_temp = dict_ocls_temp -> next_dict_cls;
```

```
     }
#ifdef EnExFlag
  printf("Exit in_dictionary false\n");
#endif

  fprintf(stderr, "%s%s\n", ocname, " does not exist in the dictionary!..");
  exit(1);
  return NULL;
}/* end of in_dictionary */


/****************************************************************************
funtion name: in_class(attrname)
return type: struct dict_attr_node
purpose: This function determines if an attribute name, given as an arguement,
         is in the current class.  If so, it returns the pointer to that
         attribute.
****************************************************************************/
struct dict_attr_node *in_class(attrname)
char  attrname[ANLength + 1];
{
  struct dict_attr_node  *dict_attr_temp;

#ifdef EnExFlag
  printf("Enter in_class\n");
#endif

  dict_attr_temp = dict_attr_ptr;
  while(dict_attr_temp) {
       if( !strcmp(dict_attr_temp -> dict_attr_name, attrname) ){
#ifdef EnExFlag
            printf("Exit in_class true\n");
#endif
            return (dict_attr_temp);}
       dict_attr_temp = dict_attr_temp -> next_dict_attr;
     }

#ifdef EnExFlag
  printf("Exit in_class false\n");
#endif

  return NULL;
}/* end of in_class */


/****************************************************************************
funtion name: check_inherit()
return type: struct dict_attr_node
purpose: This function determines if there is an inheritance relation in
         the current class. If so, then it returns a pointer to that
         attribute row.
****************************************************************************/
struct dict_attr_node *check_inherit()
{
  struct dict_attr_node *dict_att_temp;

#ifdef EnExFlag
  printf("Enter check_inherit\n");
#endif

  dict_att_temp = dict_attr_ptr;
  while(dict_att_temp) {
        if( !strcmp(dict_att_temp -> dict_ref_type, "inherit") ){
#ifdef EnExFlag
            printf("Exit check_inherit true\n");
#endif
            return (dict_att_temp);}

        dict_att_temp = dict_att_temp->next_dict_attr;
    }

#ifdef EnExFlag
  printf("Exit check_inherit false\n");
#endif

  return NULL;
}/*end of check_inherit */


/*****************************************************************************
funtion name: check_cover()
return type: struct dict_attr_node
purpose: This function determines if there is a cover relation in
         the current class. If so, then it returns a pointer to that
         attribute row.
*****************************************************************************/
struct dict_attr_node *check_cover()
{
  struct dict_attr_node *dict_att_temp;

#ifdef EnExFlag
  printf("Enter check_cover\n");
#endif

  dict_att_temp = dict_attr_ptr;
  while(dict_att_temp) {
        if( !strcmp(dict_att_temp -> dict_ref_type, "cover") ){
#ifdef EnExFlag
            printf("Exit check_cover true\n");
#endif
            return (dict_att_temp);}

        dict_att_temp = dict_att_temp->next_dict_attr;
    }

#ifdef EnExFlag
  printf("Exit check_cover false\n");
#endif
```

```
  return NULL;
}/*end of check_cover */


/*****************************************************************************
funtion name: in_symtab(varname)
return type: struct symtabl
purpose: This function determines if a variable is in the symbol table.
         If so, it returns a pointer to that variable.
*****************************************************************************/
struct symtabl *in_symtab(varname)
char *varname;
{
  struct symtabl *sptr ;

#ifdef EnExFlag
  printf("Enter in_symtab\n");
#endif

  for(sptr = symtabl; sptr < &symtabl[SIZE]; sptr++) {
      if (sptr->name && !strcmp(sptr->name, varname)){
#ifdef EnExFlag
          printf("Exit in_symtab true\n");
#endif
          return sptr;}
     }

#ifdef EnExFlag
  printf("Exit in_symtab false\n");
#endif

  return NULL;
}/* end of in_symtabl */


/*****************************************************************************
funtion name: convert_find_statement ()
return type: void
purpose: This function converts find_one or find_many operations to there
         equivalent ABDL queries.
*****************************************************************************/
void convert_find_statement ()
{
  int r, logical_opar;
  struct symtabl * symtab_ptr;
  char temp_result_arr[6][3];
  char text[20];

#ifdef EnExFlag
  printf("Enter convert_find_statement\n");
#endif

   /* Type checking for find_one and find_many */
  symtab_ptr = in_symtab(intmed_ptr -> result);
  if (symtab_ptr){
       if( (!strcmp(intmed_ptr -> opar, "find_one")) && (!strcmp(symtab_ptr -
> type, "obj_set"))){
            fprintf(stderr, "%s%s\n",intmed_ptr -> result," should be
obj_ref!..");
            exit(1);
       }
      if( (!strcmp(intmed_ptr -> opar, "find_many")) && (!strcmp(symtab_ptr ->
type, "obj_ref"))){
            fprintf(stderr, "%s%s\n",intmed_ptr -> result," should be
obj_set!..");
            exit(1);
       }
    }

  else {
     fprintf(stderr, "%s%s\n",intmed_ptr -> result, " is an undeclared
variable!..");
     exit(1);
    }/*end of type checking */

 symtab_ptr->clsname = strdup(intmed_ptr->arg1); /*keeps the related class for
each variable */

 r = 0;
 strcpy(reftabl_arr[s], intmed_ptr->arg1);  /*first table to be searched*/
 while(strcmp(intmed_ptr -> result, "0")) {
       intmed_ptr = intmed_ptr -> next;
       strcpy(text, intmed_ptr->opar);
/*branch for the expression part of a find_one or find_many statement*/
       switch (text[0]) {
            case ('a') : { logical_opar = 1;
                           strcpy(temp_result_arr[r],
convert_logical_statement(temp_result_arr,

  r, logical_opar));
                           r++;
                           break;}
            case ('o') : { logical_opar = 0;
                           strcpy(temp_result_arr[r],
convert_logical_statement(temp_result_arr,

  r, logical_opar));
                           r++;
                           break;}
            case ('c') : { strcpy(temp_result_arr[r],
convert_contains_statement());
                           r++;
                           a = 0;
                           s = 0;
                           break;}
            default : {  strcpy(temp_result_arr[r], convert_rel_expr());
                         r++;
```

```
                        a = 0;
                        s = 0;
                        break;}
         }/*end of switch statement */
     }/* end of while statement */
   /*This ensures that the final assignment is to the user defined variable*/
  if (subsflag){
      strcpy(abdlque_current_node->prior->prior->text, final_result); }
  else {
     strcpy(abdlque_current_node->prior->text, final_result);
   }

#ifdef EnExFlag
 printf("Exit convert_find_statement\n");
#endif

}/*end of find_statement */


/******************************************************************************
funtion name: convert_logical_statement(temp_arr, index, logopar)
return type: char *
purpose: This function handles the incremental growth of logical arguements.
         It reads the variable names and their order in the intermediate
         table and then matches it to one of the cases below. Logical
         arguements are limited to the size of the array "temp_arr" in the
         file intmed_tabl.c. The following illustration should clarify the
         meaning of the cases below.
```





```
*************************************************************************/
char *convert_logical_statement(temp_arr, index, logopar)
char temp_arr[6][3];
int index;
int logopar;
{
   int left, right;

#ifdef EnExFlag
   printf("%s\n", "Enter convert_logical_statement");
#endif


   left = 0;
   right = 0;
   --index;

   switch (index) {
       case(1) : { left = 0;
                   right =1;
                   break;}
       case(3) : { left = 2;
                   right = 3;
                   break;}
       case(4) : { left = 3;
                   right = 4;
                   break;}
       case(5) : { left = 2;
                   right = 5;
```

```
                    break;}
      }

   construct_retrieve_logical(objset_arr[p++], temp_arr[left],
temp_arr[right], logopar);
   subsflag = 0;

#ifdef EnExFlag
   printf("%s\n", "Exit convert_logical_statement");
#endif

   return (objset_arr[p-1]);

}/* end of convert_logical_statement */


/**************************************************************************
funtion name: construct_retrieve_logical(varname, leftvar, rightvar, logflag)
return type: void
purpose: This function handles the interpretation of logical operators.
**************************************************************************/
void construct_retrieve_logical(varname, leftvar, rightvar, logflag)
char varname[ANLength + 1];
char leftvar[ANLength + 1];
char rightvar[ANLength + 1];
int logflag;
{
  char temptext[80];

#ifdef EnExFlag
  printf("%s\n", "Enter construct_retrieve_logical");
#endif


  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "&");
  strcat(temptext, varname);
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  if (abdlque_pri_node)
      abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

  abdlque_current_node = abdlque_node_alloc();
  switch(logflag) {
     case(0): { strcpy(temptext, "+");
                break;}
     case(1): { strcpy(temptext, "*");
                break;}
     case(2): { strcpy(temptext, "^");
                break;}
   }
  strcat(temptext, leftvar);
  strcat(temptext, ",");
  strcat(temptext, rightvar);
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
  printf("%s\n", "Exit construct_retrieve_logical");
#endif


}/*end of construct_retrieve_logical */


/****************************************************************************
funtion name: convert_rel_expr()
return type: char  *
purpose: This function handles the interpretation of relational operators.
****************************************************************************/
char *convert_rel_expr()
{
  struct dict_attr_node *attr_ptr;
  char tempvar[10];
  char tem_val[10];

  int N = 0; /* keeps the size of the argument1 */
  int i = 0;
  int k = 0;
  int inherit_cnt;

#ifdef EnExFlag
  printf("Enter convert_rel_expr\n");
#endif


  for(i = 0; i< 5; i++)
      strcpy(attr_arr[i], "");

  a = 0;
  N = strlen(intmed_ptr ->arg1);
  for(i = 0; i< N; i++){          /* tokenize composite attribute name */
     if((intmed_ptr->arg1)[i] == '.') {
        attr_arr[a][k] = '\0';  /* end of attribute string */
        a++;
        k = 0;
        continue;
       }
     attr_arr[a][k++] = (intmed_ptr->arg1)[i];
  }
  attr_arr[a][k] = '\0';
  subsflag = 0;
  a = 0;
  if (in_dictionary(reftabl_arr[s])){
      attr_ptr = in_class(attr_arr[a]);
      if(attr_ptr){
         if(!strcmp(attr_ptr->dict_ref_type, "")){
            if( in_symtab( intmed_ptr->arg2 ))
                subsflag = 1;
            construct_retrieve_gen(reftabl_arr[s],
attr_arr[a],objset_arr[p++], intmed_ptr->opar,
                                  intmed_ptr->arg2);
            return(objset_arr[p-1]);
              }
         else {
           if(!strcmp(attr_ptr->dict_ref_type, "ref")) {
              ++s;
              ++a;
              strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
              strcpy(tempvar, construct_retrieve_ref());
              --s;
              --a; /*new*/
              subsflag = 1;
              construct_retrieve_gen(reftabl_arr[s],attr_arr[a],
                                    objset_arr[p++], equal, tempvar);
              return (objset_arr[p-1]);
            }
          }
        }
      else {
        attr_ptr = check_inherit();
        if(attr_ptr) {
          inherit_cnt = 0;
          do {
          ++inherit_cnt;
          ++s;
          strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);

          if (in_dictionary(reftabl_arr[s])) {
              attr_ptr = in_class(attr_arr[a]);
              if (attr_ptr) {
                  if(!strcmp(attr_ptr->dict_ref_type, "")) {
                      construct_retrieve_gen(reftabl_arr[s], attr_arr[a],
objset_arr[p++],
                                          intmed_ptr->opar, intmed_ptr-
>arg2);
                      strcpy(reftabl_arr[s], "");
                      s-= inherit_cnt;
                      return(objset_arr[p-1]);

                    }
                  else { /*inherit exists but value is not primitive */
                          if(!strcmp(attr_ptr->dict_ref_type, "ref")) {
                             ++s;
                             strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
                             ++a;
                             strcpy(tem_val, construct_retrieve_ref());
                             strcpy(reftabl_arr[s], "");
                             --s;
```

```
                              --a;
                              subsflag = 1;
                              construct_retrieve_gen(reftabl_arr[s],
attr_arr[a],
                                                     objset_arr[p++], equal,
objset_arr[p-2]);
                              s-= inheritcnt;

                              return(objset_arr[p-1]);

                           }
                     }
                   }

              }
           }while(attr_ptr = check_inherit());
           }
           else {
              fprintf(stderr, "%s%s\n", attr_ptr[0], " is not a valid attribute
name!");
              exit(1);
           }
         } /* for else */
   }

#ifdef EnExFlag
   printf("Exit convert_rel_expr\n");
#endif


}/* end of convert_rel_expr */


/*****************************************************************************
funtion name: construct_retrieve_gen(reftabl, attrname, arg1, relexpr, arg2)
return type: void
purpose: This function constructs ABDL retrieve operations once find_one/
        find_many operations have been broken down to their individual parts.
*****************************************************************************/
void construct_retrieve_gen(reftabl, attrname, arg1, relexpr, arg2)
char reftabl[RTLength + 1];
char attrname[ANLength + 1];
char arg1[10];
char relexpr[3];
char arg2[10];

{
  struct intmed_node *temp;
  char temptext[80];
  char ref_temp[RTLength + 1];

#ifdef EnExFlag
  printf("Enter construct_retrieve_gen\n");
#endif
```

```
  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "&");
  strcat(temptext, arg1);
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  if (abdlque_pri_node)
      abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

  if (subsflag) {
     abdlque_current_node = abdlque_node_alloc();
     strcpy(temptext, "~");
     strcat(temptext, arg2);
     strcpy(abdlque_current_node->text, temptext);
     abdlque_current_node->prior = abdlque_pri_node;
     abdlque_pri_node->next = abdlque_current_node;
     abdlque_pri_node = abdlque_current_node;
  }

  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "[RETRIEVE((TEMP=");
  strcpy(ref_temp, reftabl);
  strcat(temptext, converttoclassname(ref_temp));
  strcat(temptext, ")and(");
  strcpy(ref_temp, attrname);
  strcat(temptext, converttoupper(ref_temp));
  strcat(temptext, relexpr);
  strcat(temptext, arg2);
  strcat(temptext, "))(OID)]");
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;


#ifdef EnExFlag
  printf("Exit construct_retrieve_gen\n");
#endif


}/* end of construct_retrieve_gen */


/*****************************************************************************
funtion name: converttoupper()
return type: char *
purpose: This function takes a string as an input an converts it all to upper
         case letters.
*****************************************************************************/
char *converttoupper(input_string)
char *input_string;
{
  char convert_string[40];
  int i;

  strcpy(convert_string, input_string);
  for (i=0; convert_string[i]; i++)
       convert_string[i] = toupper(convert_string[i]);
  return convert_string;
}/* end of converttoupper */
/****************************************************************************
funtion name: converttoclassname(input_string)
return type: char *
purpose: This function converts the first character of a string into an upper
         case letter
****************************************************************************/
char *converttoclassname(input_string)
char input_string[ANLength + 1];
{
  input_string[0] = toupper(input_string[0]);

  return input_string;

}/* end of converttoclassname */


/****************************************************************************
funtion name: converttolower(input_string)
return type: char *
purpose: This function takes a string as an input an converts it all to upper
         case letters.
****************************************************************************/
char *converttolower(input_string)
char *input_string;
{
  char convert_string[40];
  int i;

  strcpy(convert_string, input_string);
  for (i=0; convert_string[i]; i++)
       convert_string[i] = tolower(convert_string[i]);
  return convert_string;
}/* end of converttolower*/


/****************************************************************************
funtion name: construct_retrieve_ref()
return type: char *
purpose: This function is called by convert_rel_expression when the value of
         the Rel Type column of the data dictionary for the first attribute is
         ref.
****************************************************************************/
char *construct_retrieve_ref()
{
  struct dict_attr_node *att_ptr;
  char temp_var[3];
```

```
#ifdef EnExFlag
  printf("Enter construct_retrieve_ref\n");
#endif


  if (in_dictionary(reftabl_arr[s])) {
      att_ptr = in_class(attr_arr[a]);
      if (att_ptr) {
          if (!strcmp(att_ptr->dict_ref_type, "")) {
               construct_retrieve_gen(reftabl_arr[s], attr_arr[a],
objset_arr[p++], intmed_ptr->opar,
                                       intmed_ptr->arg2);
               return (objset_arr[p-1]);
           }
          if (!strcmp(att_ptr->dict_ref_type, "ref")){
              ++s;
              ++a;
              strcpy(reftabl_arr[s], att_ptr->dict_ref_table);
              strcpy(temp_var, construct_retrieve_ref());
              --s;
              --a;
              subsflag = 1;
              construct_retrieve_gen(reftabl_arr[s], attr_arr[a],
objset_arr[p++], equal,
                                       temp_var);
              return (objset_arr[p-1]);
            }

        }
      else {
         att_ptr = check_inherit();
         if (att_ptr){
             ++s;
             strcpy(reftabl_arr[s], att_ptr->dict_ref_table);
             strcpy(temp_var, construct_retrieve_ref());
             --s;
             return(temp_var);

             }
      }
  }

#ifdef EnExFlag
 printf("Exit construct_retrive_ref\n");
#endif


}/*end  of construct_retrieve_ref */


/***************************************************************************
funtion name: construct_retrieve_by (reftabl, attrname, arg1, relexpr, arg2,
               targetlist, sortvar)
```

```
return type: void
purpose: This function handles the limitation of the ABDB when the target list
         of retrieve statements does not contain an OID.
*************************************************************************/
void construct_retrieve_by (reftabl, attrname, arg1, relexpr, arg2,
targetlist, sortvar)
char reftabl[ANLength + 1];
char attrname[ANLength + 1];
char arg1[3];
char relexpr[3];
char arg2[ANLength +1];
char targetlist[80];
char sortvar[ANLength + 1];
{
  char temptext[100];
  char ref_temp[60];

#ifdef EnExFlag
  printf("Enter construct_retrieve_by\n");
#endif


  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "&");
  strcat(temptext, arg1);
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  if (abdlque_pri_node)
      abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

   if (subsflag) {
     abdlque_current_node = abdlque_node_alloc();
     strcpy(temptext, "~");
     strcat(temptext, arg2);
     strcpy(abdlque_current_node->text, temptext);
     abdlque_current_node->prior = abdlque_pri_node;
     abdlque_pri_node->next = abdlque_current_node;
     abdlque_pri_node = abdlque_current_node;
  }

  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "[RETRIEVE((TEMP=");
  strcpy(ref_temp, reftabl);
  strcat(temptext, converttoclassname(ref_temp));
  strcat(temptext, ")and(");
  strcpy(ref_temp, attrname);
  strcat(temptext, converttoupper(ref_temp));
  strcat(temptext, relexpr);
  strcat(temptext, arg2);
  strcat(temptext, "))(");
  strcpy(ref_temp, targetlist);
  strcat(temptext, converttoupper(ref_temp));
  strcat(temptext, ")BY ");
  strcat(temptext, converttoupper(sortvar));
  strcat(temptext, "]");
  strcpy(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  abdlque_pri_node->next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
  printf("Exit construct_retrieve_by\n");
#endif


}/* end of construct_retrieve_by*/


/**************************************************************************
funtion name: convert_contains_statement()
return type: char *
purpose: This function is called by convert_find_statement to handle the
         contains branch.  It simplifies the contains operation into an ABDL
         retrieve operation.  Specifically for covering, set_of and inverse_of
         relations.
**************************************************************************/
char *convert_contains_statement()
{
  struct dict_attr_node *attr_ptr;
  struct symtabl  *symtab_ptr;
  char temptext[ANLength + 1];
  char temp1[ANLength + 1];
  char temp2[60];
  char settype_arr[2][20];
  int length, i, k, l, logopar;

#ifdef EnExFlag
  printf("Enter convert_contains_statement\n");
#endif


/*This section of the code specifically handles covering type1*/
  if (!strcmp(intmed_ptr->arg1, "it")){
      if(in_dictionary(reftabl_arr[0])){
         attr_ptr = check_cover();
         if(attr_ptr){
            ++s;
            strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
            if(in_dictionary(reftabl_arr[s])){
               strcpy(temp1, "oid_");
               symtab_ptr = in_symtab(intmed_ptr->arg2);
               if(!symtab_ptr){
                  fprintf(stderr, "%s%s\n", intmed_ptr->arg2, " is an
undeclared variable!..");
                  exit(1);
                        }
               else{
                    strcat(temp1, symtab_ptr->clsname);
                    attr_ptr = in_class(temp1);
                    if(attr_ptr){
                       strcpy(temp2, "OID_");
                       strcpy(temptext, reftabl_arr[s-1]);
                       strcat(temp2, converttoupper(temptext));
                       converttoupper(temp1);
                       subsflag = 1;
                       construct_retrieve_by(reftabl_arr[s], temp1,
objset_arr[p++],
                                                 equal, intmed_ptr->arg2, temp2,
temp2);
                       logopar = 2;
                       construct_retrieve_logical(objset_arr[p++], objset_arr[p-
2],
                                                      intmed_ptr->arg2, logopar);
                       strcpy(reftabl_arr[s], "");
                       --s;
                       subsflag = 0;
                       return(objset_arr[p-1]);
                     }
                  }
                   }
                 }
          }
      }

   if (!strcmp(intmed_ptr->arg2, "it")){
        printf("%s\n","This type of covering type2 relationship will be
implemented later!");
        exit(1);
      }
   else{
      strcpy(attr_arr[0], intmed_ptr->arg1);
      if(in_dictionary(reftabl_arr[s])){
         attr_ptr = in_class(attr_arr[0]);
         if (attr_ptr){
           /*This part handles set_of relationship*/
            if(!strcmp(attr_ptr->dict_attr_type, "set_of")){
               ++s;
               strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
                   if(in_dictionary(reftabl_arr[s])){
                 strcpy(temp1, "oid_");
                 symtab_ptr = in_symtab(intmed_ptr->arg2);/*error checking
can be written */
                 strcat(temp1, symtab_ptr->clsname);
                 attr_ptr = in_class(temp1);
                 if (attr_ptr) {
                     strcpy(temp2, "OID_");
                     strcpy(temptext, reftabl_arr[s-1]);
                     strcat(temp2, converttoupper(temptext));
                     converttoupper(temp1);
                     subsflag = 1;
                     construct_retrieve_by(reftabl_arr[s], temp1,
objset_arr[p++], equal,
                                                  intmed_ptr->arg2, temp2, temp2);
                        strcpy(reftabl_arr[s], "");
                        --s;
                        return (objset_arr[p-1]);
                      }
                          }
                   }
             else{
            /*This part handles inverse_of relations*/
               if(!strcmp(attr_ptr->dict_attr_type, "inverse_of")){
                k = 0;
                l = 0;
                length = strlen (attr_ptr->dict_ref_table);
                for(i = 0; i< length; i++) {
                    if((attr_ptr->dict_ref_table)[i] == '.') {
                        settype_arr[k][l] = '\0';
                        k++;
                        l = 0;
                        continue;
                     }
                    settype_arr[k][l++] = (attr_ptr->dict_ref_table)[i];
                 } /* end of for_statement */
                settype_arr[k][l] = '\0';

                if(in_dictionary(settype_arr[0])) {
                   attr_ptr = in_class(settype_arr[1]);
                   if(attr_ptr) {
                      ++s;
                      strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
                      if(in_dictionary(reftabl_arr[s])) {
                         strcpy(temp1, "oid_");
                         symtab_ptr = in_symtab(intmed_ptr->arg2);/*error
checking can be written */
                         strcat(temp1, symtab_ptr->clsname);
                         attr_ptr = in_class(temp1);
                         if (attr_ptr) {
                            strcpy(temp2, "OID_");
                            strcpy(temptext, reftabl_arr[s-1]);
                            strcat(temp2, converttoupper(temptext));
                            converttoupper(temp1);
                            subsflag = 1;
                            construct_retrieve_by(reftabl_arr[s], temp1,
objset_arr[p++], equal,
                                                       intmed_ptr->arg2, temp2,
temp2);
                            strcpy(reftabl_arr[s], "");
                            --s;
                            return (objset_arr[p-1]);
                          }
                              }
                            }
                         }
                 }
                 }
              }

           }
       }

#ifdef EnExFlag
  printf("Exit convert_contains statement\n");
#endif


} /* end of convert_contains_statement */


/*****************************************************************************
funtion name: createDict()
return type: struct obj_dbid_node *
purpose: This function reads the text file of the data dictionary and
         converts it to a linked  list structure.
*****************************************************************************/
struct obj_dbid_node *createDict() {

   struct obj_dbid_node *db_pointer,
                        *obj_dbid_node_alloc();
   struct dict_ocls_node *dict_cls_ptr, *temp_cls, *current_cls,
                         *dict_ocls_node_alloc();
   struct dict_attr_node *dict_attr_ptr, *temp_attr, *current_attr,
                         *dict_attr_node_alloc();
   FILE *sourcefile;
   char str[20], tempstr[20];
   int number, i;

#ifdef EnExFlag
   printf("Enter createDict\n");
#endif


   sourcefile = fopen( "RECEPT.dict.frm", "r");

   db_pointer = obj_dbid_node_alloc();


   fgets(str, 20, sourcefile);
        for(number =0; str[number] != '\n'; number++);
            str[number] = '\0';

   strcpy(db_pointer->odn_name, converttolower(str));

   do {

     fgets(str, 20, sourcefile);
        for(number =0; str[number] != '\n'; number++);
            str[number] = '\0';
```

```
if(str[0] == '@') {
  dict_cls_ptr = dict_ocls_node_alloc();

  fgets(str, 20, sourcefile);
   for(number =0; str[number] != '\n'; number++);
       str[number] = '\0';

  strcpy(dict_cls_ptr->dict_ocn_name, converttolower(str));
  if(db_pointer->odn_first_dict_cls == NULL) {
     db_pointer->odn_first_dict_cls = dict_cls_ptr;
     db_pointer->odn_curr_dict_cls = dict_cls_ptr;
       }
  else {
     temp_cls = db_pointer->odn_first_dict_cls;

     while(temp_cls != NULL) {
        current_cls = temp_cls;
        temp_cls = temp_cls->next_dict_cls;
          }
     current_cls->next_dict_cls = dict_cls_ptr;
     db_pointer->odn_curr_dict_cls = dict_cls_ptr;
       }


}



   if(str[0] == '#') {
    dict_attr_ptr = dict_attr_node_alloc();

       if (db_pointer->odn_curr_dict_cls->dict_first_attr == NULL) {
      db_pointer->odn_curr_dict_cls->dict_first_attr = dict_attr_ptr;
      db_pointer->odn_curr_dict_cls->dict_curr_attr = dict_attr_ptr;
        }
       else {
          temp_attr = db_pointer->odn_curr_dict_cls->dict_first_attr;

          while(temp_attr != NULL) {
             current_attr = temp_attr;
         temp_attr = temp_attr->next_dict_attr;
            }

          current_attr->next_dict_attr = dict_attr_ptr;
          db_pointer->odn_curr_dict_cls->dict_curr_attr = dict_attr_ptr;
        }

    current_attr = db_pointer->odn_curr_dict_cls->dict_curr_attr;


    fgets(str, 20, sourcefile);
    for(number =0; str[number] != '\n'; number++);
    str[number] = '\0';
```

```
        strcpy(current_attr->dict_attr_name, converttolower(str));


        fgets(str, 20, sourcefile);
        for(number = 0; str[number] != '\n'; number++);
        str[number] = '\0';
            strcpy(current_attr->dict_attr_type, converttolower(str));



        fgets(str, 20, sourcefile);
        for(number = 0; str[number] != '\n'; number++);
        str[number] = '\0';
        strcpy(current_attr->dict_ref_table, converttolower(str));


        fgets(str, 20, sourcefile);
        for(number = 0; str[number] != '\n'; number++);
        str[number] = '\0';
        strcpy(current_attr->dict_ref_type, converttolower(str));


     }


     } while (str[0] != '$');

#ifdef EnExFlag
   printf("Exit createDict\n");
#endif

   return db_pointer;

 }


/*****************************************************************************
funtion name: display_abdl()
return type: void
purpose: This function can be used for debugging purposes to display the
         output of the query_constructor().
*****************************************************************************/
void display_abdl()
{
  struct abdlque_node *temp;

  temp = abdlque_first_node;
  printf("\n%s\n", "pseudocode for real-time monitor");
  while(temp){
       printf("%s\n", temp->text);
       temp = temp -> next;
  }

}
```

```
/*****************************************************************************
funtion name: add_varlist()
return type: void
purpose: This function puts all the user defined and query_constructor defined
         variables as the first and second lines of the output.
*****************************************************************************/
void add_varlist()
{
  char temptext[100];
  struct symtabl  *symptr;
  int i;
  int comma_flag = 0;

#ifdef EnExFlag
  printf("%s\n", "Enter add_varlist");
#endif


  abdlque_current_node = abdlque_node_alloc();
  strcpy (temptext, "%");
  for(symptr = symtabl;symptr->name ; symptr++){
      if(!strcmp(symptr->type, "objr")){
          if ( comma_flag )
               strcat(temptext, ",");
        comma_flag = 1;
        strcat(temptext, symptr->name);
       }
     }

  for( i=0; i< f; i++){
       if ( comma_flag )
          strcat(temptext, ",");
       comma_flag = 1;
       strcat(temptext, objref_arr[i]);
   }

  comma_flag = 0;
  strcat(abdlque_current_node->text, temptext);
  abdlque_pri_node = abdlque_current_node;

  abdlque_current_node = abdlque_node_alloc();
  strcpy (temptext, "@");
  for(symptr = symtabl;symptr->name ; symptr++){
      if(!strcmp(symptr->type, "objs")){
         if ( comma_flag )
              strcat(temptext, ",");
         comma_flag = 1;
         strcat(temptext, symptr->name);
       }

     }
  for( i=0; i< p; i++){
       if ( comma_flag )
           strcat(temptext, ",");
       comma_flag = 1;
       strcat(temptext, objset_arr[i]);
   }
  comma_flag = 0;
  strcat(abdlque_current_node->text, temptext);
  abdlque_current_node->prior = abdlque_pri_node;
  abdlque_pri_node->next = abdlque_current_node;
  abdlque_current_node->next = abdlque_first_node;
  abdlque_first_node->prior = abdlque_current_node;
  abdlque_first_node = abdlque_pri_node;

#ifdef EnExFlag
  printf("%s\n", "Exit add_varlist");
#endif

} /* end of add_varlist  */


/*******************************************************************************
funtion name: convert_loop_statement()
return type: void
purpose: This function handles for_loop operations
*******************************************************************************/
void convert_loop_statement()
{
 struct symtabl *symptr1;
 struct symtabl *symptr2;
 char temptext[80];

#ifdef EnExFlag
 printf("%s\n", "Enter convert_loop_statement");
#endif

 /*type checking and assignment of  the classname to the counter variable */

 if(!strcmp(intmed_ptr->opar, "jump")){
 symptr1 = in_symtab(intmed_ptr->arg1);
 symptr2 = in_symtab(intmed_ptr->arg2);
 if(!symptr1){
    fprintf(stderr, "%s%s\n",intmed_ptr->arg1, " is undeclared variable!...");
    exit(1);
   }
 else {
   if(!symptr2){
        fprintf(stderr, "%s%s\n",intmed_ptr->arg2, " is undeclared
variable!...");
        exit(1);
      }
  }

  (symptr1->clsname) = strdup(symptr2->clsname);
```

```
}

 abdlque_current_node = abdlque_node_alloc();
 if(!strcmp(intmed_ptr->opar, "jump")){
    strcpy(temptext, "$");
    strcat(temptext, intmed_ptr->arg1);
    strcat(temptext, ",");
    strcat(temptext, intmed_ptr->arg2);
  }
 else{
    strcpy(temptext, "!");
  }
 strcpy(abdlque_current_node->text, temptext);
 abdlque_current_node->prior = abdlque_pri_node;
 if(abdlque_pri_node)
    abdlque_pri_node->next = abdlque_current_node;
 abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
 printf("%s\n", "Exit convert_loop_statement");
#endif


}/* end of convert_loop_statement */


/*****************************************************************************
funtion name: convert_display_statement()
return type: void
purpose: This function converts display operations to their ABDL equivalent
         queries.
*****************************************************************************/
void convert_display_statement()
{
 struct symtabl  *symptr;
 struct dict_attr_node  *attr_ptr;
 struct dict_attr_node  *attr_temp;

 char display_arr[6][MAXPATHLEN][ANLength + 1];/* Not more than six arguments!
*/
 char attrlist[100];
 char first_attr[ANLength + 1];
 char attrname[ANLength + 1];

 int length, rf, a, k, i, b, u;

#ifdef EnExFlag
 printf("%s\n", "Enter convert_display_statement");
#endif

 strcpy(attrname, "oid");
 b = 0;
 while(!strcmp(intmed_ptr->opar, "display")){
    a = 0;
    k = 0;
    length = strlen(intmed_ptr->arg1);
    for(i=0; i< length; i++){
        if((intmed_ptr->arg1)[i] == '.'){
            display_arr[b][a][k] = '\0';
            ++a;
            k = 0;
            continue;
          }
        display_arr[b][a][k++] = (intmed_ptr->arg1)[i];
      }
    display_arr[b][a][k] = '\0';

    if(intmed_ptr->arg2){
       ++b;
       a = 0;
       k = 0;
       length = strlen(intmed_ptr->arg2);
       for(i=0; i< length; i++){
           if((intmed_ptr->arg2)[i] == '.'){
            display_arr[b][a][k] = '\0';
            ++a;
            k = 0;
            continue;
          }
         display_arr[b][a][k++] = (intmed_ptr->arg2)[i];
       }
       display_arr[b][a][k] = '\0';
     }

    ++b;
    if(!strcmp(intmed_ptr->result, "1"))
       intmed_ptr = intmed_ptr->next;
    else
       break;
  }
 s = 0;
 subsflag = 0;
 for(i=0; i< b; i++){
     symptr = in_symtab(display_arr[i][0]);
     if(symptr)
        strcpy(reftabl_arr[s], symptr->clsname);
     else{
        fprintf(stderr,"%s%s\n", display_arr[i][0], " is undeclared
variable!..");
        exit(1);
      }
     rf = 1;
     do {

     if(in_dictionary(reftabl_arr[s])){
        attr_ptr = in_class(display_arr[i][rf]);
        if(attr_ptr){
          if(!strcmp(attr_ptr->dict_ref_type, "")){
             strcpy(first_attr,display_arr[i][rf]);
             strcpy(attrlist,first_attr );
             for(u=(i+1); u< b; u++) { /* checks the following arguments to
                                        if they are also in the same table*/
                 if(display_arr[u][rf]){
                    attr_temp = in_class(display_arr[u][rf]);
                    if(attr_temp){
                       if(!strcmp(attr_temp->dict_ref_type, "")){
                          strcat(attrlist, ",");
                          strcat(attrlist, display_arr[u][rf]);
                          ++i;
                                }
                       else
                         break; /* if not in the same table then get out of
                                the for loop*/
                             }
                          }
                       }/*end of for_statement */
             construct_retrieve_display(reftabl_arr[s], display_arr[i][0],
attrlist, first_attr);
             break;
           }
          else{
             if(!strcmp(attr_ptr->dict_ref_type, "ref")){
                subsflag = 1;
                construct_retrieve_by(reftabl_arr[s], attrname,
objref_arr[f++],
                                        equal, display_arr[i][0],
display_arr[i][rf],display_arr[i][rf] );
                ++rf;
                do { /*continue looking for the attribute until the primitive
                       value is found*/
                strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
                if(in_dictionary(reftabl_arr[s])){
                   attr_ptr = in_class(display_arr[i][rf]);
                   if(attr_ptr){
                      if(!strcmp(attr_ptr->dict_ref_type, "")){
                         strcpy(first_attr,display_arr[i][rf]);
                         strcpy(attrlist,first_attr );
                         for(u=(i+1); u< b; u++){
                            if(display_arr[u][rf] &&
!strcmp(display_arr[u][rf-1],display_arr[u-1][rf-1])){
                               attr_temp = in_class(display_arr[u][rf]);
                               if(attr_temp){
                                  if(!strcmp(attr_temp->dict_ref_type, "")){
                                     strcat(attrlist, ",");
                                     strcat(attrlist, display_arr[u][rf]);
                                     ++i;
                                              }
                                  else
                                    break; /*gets out of loop if preceeding
                                           the path expression is not same*/
                                }
```

```
                                                    }
                                        else
                                          break; /* gets out of loop if the current
                                                 attribute is not in the same table*/
                                               }/*end of for_statement */
                                      construct_retrieve_display(reftabl_arr[s],
objref_arr[f-1],
                                                                attrlist, first_attr);
                                      break; /* gets out of loop when you find the
                                             primitive value (in only ref case)*/
                                           }
                                   else{
                                      if(!strcmp(attr_ptr->dict_ref_type, "ref")){
                                         subsflag = 1;
                                         construct_retrieve_by(reftabl_arr[s],attrname ,
objref_arr[f++],
                                                               equal, objref_arr[f-
2],display_arr[i][rf],display_arr[i][rf]);
                                         ++rf;
                                                }
                                           }
                                        }
                             else  /* second attribute is not in the table */
                                attr_ptr = check_inherit();
                                      }
                                      }while(attr_ptr);
                         break;/* gets out of loop when you find the
                                            primitive value (inherit case)*/
                              }
                           }
                        }

        else{ /* first attribute is not in the table */
           attr_ptr = check_inherit();
           if(attr_ptr)
              strcpy(reftabl_arr[s], attr_ptr->dict_ref_table);
        }
      }
      }while(attr_ptr);
     continue;
    }/* end of for_statement */

#ifdef EnExFlag
 printf("%s\n", "Exit convert_display_statement");
#endif

}/*end of convert_display_statement*/

/********************************************************************************
funtion name: construct_retrieve_display()
return type: void
purpose: This function constructs display operations in ABDL queries.
*******************************************************************************/
void construct_retrieve_display(reftabl, varname, targetlist, sortvar)
```

```
char reftabl[ANLength + 1];
char varname[ANLength + 1];
char targetlist[80];
char sortvar[ANLength + 1];
{
 char temptext[100];
 char ref_temp[60];

#ifdef EnExFlag
 printf("%s\n", "Enter construct_retrieve_display");
#endif


 abdlque_current_node = abdlque_node_alloc();
 strcpy(temptext, "~");
 strcat(temptext, varname);
 strcpy(abdlque_current_node->text, temptext);
 abdlque_current_node->prior = abdlque_pri_node;
 if(abdlque_pri_node)
    abdlque_pri_node->next = abdlque_current_node;
 abdlque_pri_node = abdlque_current_node;

 abdlque_current_node = abdlque_node_alloc();
 strcpy(temptext, "[ORETRIEVE((TEMP=");
 strcpy(ref_temp, reftabl);
 strcat(temptext, converttoclassname(ref_temp));
 strcat(temptext, ")and(OID=");
 strcat(temptext, varname);
 strcat(temptext, "))(");
 strcat(temptext, converttoupper(targetlist));
 strcat(temptext, ")BY ");
 strcat(temptext, converttoupper(sortvar));
 strcat(temptext, "]");
 strcpy(abdlque_current_node->text, temptext);
 abdlque_current_node->prior = abdlque_pri_node;
 abdlque_pri_node->next = abdlque_current_node;
 abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
 printf("%s\n", "Exit construct_retrieve_display");
#endif

}/*end of construct_retrieve_display */

/*****************************************************************************
funtion name: convert_assign_statement()
return type: void
purpose: This function converts assign operations into their ABDL equivalent
         queries.

*****************************************************************************/
void convert_assign_statement()
{
   struct dict_attr_node *attr_ptr;
   struct symtabl *symptr;
   char path_expr[MAXPATHLEN][ANLength + 1];
   char temp_oid[10];
   int N = 0;              /*keeps the size of argument1 */
   int i, k, rf;

#ifdef EnExFlag
   printf("Enter convert_assign_statement\n");
#endif

   a = 0;
   k = 0;
   strcpy(temp_oid, "oid");
   N = strlen(intmed_ptr -> result);
   for (i = 0; i < N; i++){         /*tokenize composite attributes*/
      if ((intmed_ptr -> result)[i] == '.'){
         path_expr[a][k] = '\0';    /*end of attribute string*/
         a++;
         k = 0;
         continue;
      }
      path_expr[a][k++] = (intmed_ptr -> result)[i];
   }
   path_expr[a][k] = '\0';

   symptr = in_symtab(path_expr[0]);
   if (!symptr) {
      fprintf(stderr,"%s%s\n",path_expr[0], " is an undeclared variable!..");
      exit(1);
   }

   s = 0;
   strcpy(reftabl_arr[s], symptr -> clsname);
   rf = 1;

   do {
   if(in_dictionary(reftabl_arr[s])){
      attr_ptr = in_class(path_expr[rf]);
       if(attr_ptr){
          if(!strcmp(attr_ptr->dict_ref_type, "")){
             construct_update(reftabl_arr[s], path_expr[0], path_expr[rf],
intmed_ptr -> arg1);
             break;
          }
          else {
             if(!strcmp(attr_ptr -> dict_ref_type, "ref")){
                subsflag = 1;
                construct_retrieve_by(reftabl_arr[s], temp_oid,
objref_arr[f++], equal,
                                         path_expr[0], path_expr[rf],
path_expr[rf]);
                ++rf;
                do {
                strcpy(reftabl_arr[s], attr_ptr -> dict_ref_table);
                if(in_dictionary(reftabl_arr[s])) {
                   attr_ptr = in_class(path_expr[rf]);
                   if(attr_ptr){
                      if(!strcmp(attr_ptr -> dict_ref_type, "")){
                         construct_update(reftabl_arr[s], objref_arr[f-1],
path_expr[rf],
                                          intmed_ptr -> arg1);
                         break;
                      }
                      else {
                         if(!strcmp(attr_ptr -> dict_ref_type, "ref")){
                            subsflag = 1;
                            construct_retrieve_by(reftabl_arr[s], temp_oid,
objref_arr[f++], equal,
                                                  objref_arr[f-2],
path_expr[rf], path_expr[rf]);
                            ++rf;
                            }
                                 }
                           }
                   else
                      attr_ptr = check_inherit();
                        }
                        }while(attr_ptr);
                break;
                 }
              }
       }
       else {
         attr_ptr = check_inherit();
         if(attr_ptr)
              strcpy(reftabl_arr[s], attr_ptr -> dict_ref_table);
       }
    }
    }while(attr_ptr);

#ifdef EnExFlag
 printf("Exit convert_assign_statement\n");
#endif

  return;

} /* end of convert_assign */

/*************************************************************************
funtion name: construct_update()
return type: void
purpose: This function constructs update ABDL queries.
*************************************************************************/
void construct_update(reftabl, oid_value, attrname, attrvalue)
char reftabl[ANLength + 1];
char oid_value[ANLength + 1];
char attrname[ANLength + 1];
char attrvalue[60];
```

```
{

  char temptext[100];
  char ref_temp[60];

#ifdef EnExFlag
  printf("Entering construct_update\n");
#endif

  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "~");
  strcat(temptext, oid_value);
  strcpy(abdlque_current_node -> text, temptext);
  abdlque_current_node -> prior = abdlque_pri_node;
  if(abdlque_pri_node)
     abdlque_pri_node -> next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

  abdlque_current_node = abdlque_node_alloc();
  strcpy(temptext, "[UPDATE((TEMP=");
  strcpy(ref_temp, reftabl);
  strcat(temptext, converttoclassname(ref_temp));
  strcat(temptext, ")and(OID=");
  strcat(temptext, oid_value);
  strcat(temptext, "))<");
  strcpy(ref_temp, attrname);
  strcat(temptext, converttoupper(ref_temp));
  strcat(temptext, "=");
  strcat(temptext, attrvalue);
  strcat(temptext, ">]");
  strcpy(abdlque_current_node -> text, temptext);
  abdlque_current_node -> prior = abdlque_pri_node;
  abdlque_pri_node -> next = abdlque_current_node;
  abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
  printf("Exiting construct_update\n");
#endif

  return;

}/* end of construct_update */

/*****************************************************************************
funtion name: convert_add_statement()
return type: void
purpose: This function converts add operations to their ABDL equivalent
         queries. (set_of, inverse_of, and cover update function)
*****************************************************************************/
void convert_add_statement()
{

   struct dict_attr_node *attr_ptr;
   struct symtabl *symptr;
   char path_expr[MAXPATHLEN][ANLength + 1];
   char temp_oid[10];
   int N = 0;              /*keeps the size of argument1 */
   int i = 0;
   int k = 0;

#ifdef EnExFlag
   printf("Enter convert_add_statement\n");
#endif

   a = 0;

   N = strlen(intmed_ptr -> arg1 );
   for (i = 0; i < N; i++){         /*tokenize composite attributes*/
      if ((intmed_ptr -> arg1)[i] == '.'){
         path_expr[a][k] = '\0';     /*end of attribute string*/
         a++;
         k = 0;
         continue;
      }
      path_expr[a][k++] = (intmed_ptr -> arg1)[i];
   }
   path_expr[a][k] = '\0';     /*end of attribute string*/
   symptr = in_symtab(path_expr[0]);
   if (!symptr) {
        fprintf(stderr,"%s%s\n",path_expr[0], " is an undeclared variable!..");
        exit(1);
   }

   s = 0;

   strcpy(reftabl_arr[s], symptr -> clsname);
   if(in_dictionary(reftabl_arr[s])){
      if(a > 0) {             /*then this is a store add */
         attr_ptr = in_class(path_expr[1]);
         if(attr_ptr){
            symptr = in_symtab(intmed_ptr -> arg2); /*look at arg2*/
            if(!symptr) {    /*if not declared then error !*/
               fprintf(stderr,"%s%s\n",intmed_ptr -> arg2,"is an undeclared
variable!");
               exit(1);
            }
            else{          /* arg2 is in the symbol table */
               ++s;
               strcpy(reftabl_arr[s],  attr_ptr -> dict_ref_table);
               if(in_dictionary(reftabl_arr[s]))
                  construct_add(reftabl_arr[s], path_expr[0], intmed_ptr ->
arg2, reftabl_arr[0]);
            }
         }
      }
      else {                  /*then this is a cover add */
         attr_ptr = check_cover();
         ++s;
```

```
          strcpy(reftabl_arr[s], attr_ptr -> dict_ref_table);
          symptr = in_symtab(intmed_ptr -> arg2);
          if(!symptr){
            fprintf(stderr,"%s%s\n",intmed_ptr -> arg2,"is an undeclared
variable!");
            exit(1);
          }
          else
             if(in_dictionary(reftabl_arr[s]))
               construct_add(reftabl_arr[s], path_expr[0], intmed_ptr -> arg2,
reftabl_arr[0]);

      }
   }
#ifdef EnExFlag
   printf("Exiting convert_add_statement\n");
#endif
}

/***********************************************************************
funtion name: construct_add()
return type: void
purpose: This function constructs add operations into their ABDL equivalent
         queries.

***********************************************************************/
void construct_add(reftable, attr_arg1, attr_arg2, arg1_cls_name)
char reftable [ANLength + 1];
char attr_arg1[ANLength + 1];
char attr_arg2[ANLength + 1];
char arg1_cls_name[ANLength + 1];
{

   char temptext[100];
   char ref_temp[60];
   struct dict_attr_node *dict_att_temp;

#ifdef EnExFlag
   printf("Entering construct_add\n");
#endif


   abdlque_current_node = abdlque_node_alloc();
   strcpy(temptext, "#");
   strcat(temptext, attr_arg1);
   strcat(temptext, ",");
   strcat(temptext, attr_arg2);
   strcpy(abdlque_current_node -> text, temptext);
   abdlque_current_node -> prior = abdlque_pri_node;
   abdlque_pri_node -> next = abdlque_current_node;
   abdlque_pri_node = abdlque_current_node;


   abdlque_current_node = abdlque_node_alloc();
   strcpy(temptext, "[AINSERT(<TEMP,");
   strcpy(ref_temp, reftable);
   strcat(temptext, converttoclassname(ref_temp));
   strcat(temptext, ">,<OID,?>,<");

   dict_att_temp = dict_attr_ptr;
   dict_att_temp = dict_att_temp -> next_dict_attr;
   strcpy(ref_temp, dict_att_temp -> dict_attr_name);
   strcat(temptext, convertoupper(ref_temp));
   strcat(temptext, ",");
   if(!strcmp(dict_att_temp -> dict_ref_table, arg1_cls_name))
     strcat(temptext, attr_arg1);
   else
     strcat(temptext, attr_arg2);

   strcat(temptext, ">,<");

   dict_att_temp = dict_att_temp -> next_dict_attr;
   strcpy(ref_temp, dict_att_temp -> dict_attr_name);
   strcat(temptext, convertoupper(ref_temp));
   strcat(temptext, ",");
   if(!strcmp(dict_att_temp -> dict_ref_table, arg1_cls_name))
     strcat(temptext, attr_arg1);
   else
     strcat(temptext, attr_arg2);

   strcat(temptext, ">)]");

   strcpy(abdlque_current_node -> text, temptext);
   abdlque_current_node -> prior = abdlque_pri_node;
   abdlque_pri_node -> next = abdlque_current_node;
   abdlque_pri_node = abdlque_current_node;

#ifdef EnExFlag
   printf("Exiting construct_add\n");
#endif

}/*end of construct_add */

/****************************************************************************
funtion name: clean_lists()
return type: void
purpose: This function frees up the allocated memory used in the intermediate
         language table.
****************************************************************************/
void cleanup_lists()
{
  struct symtabl *symptr;

#ifdef EnExFlag
 /* printf("%s\n", "Enter cleanup_lists");*/
#endif

 /*Clean up the allocated memory for intermediate table */
```

```
  while(intmed_first_node){
        free(intmed_first_node);
        intmed_first_node = intmed_first_node-> next;
     }

 /*Clean up the allocated memory for pseudocode for real-time monitor */
  while(abdlque_first_node){
        free(abdlque_first_node);
        abdlque_first_node = abdlque_first_node-> next;
     }
 /*Clean up the symbol_table entries*/

  for(symptr=symtabl; symptr->name; symptr++){
       if(symptr->name){
          free(symptr->name);
          symptr->name = NULL;}
       if(symptr->type){
          free(symptr->type);
          symptr->type = NULL;}
       if(symptr->clsname){
          free(symptr->clsname);
          symptr->clsname = NULL;}
     }
#ifdef EnExFlag
  /*printf("%s\n", "Exit cleanup_lists");*/
#endif

}

/*****************************************************************************
funtion name: input_to_rtm()
return type: void
purpose: This function creates a file and copies the newly created ABDL
         queries into it for use by the RTM.
*****************************************************************************/

void input_to_rtm()
{
 char temptext[100];
 struct abdlque_node  * temp;
 FILE *filePtr;    /* Defines a file pointer */

#ifdef EnExFlag
 /*printf("%s\n", "Enter input_to_rtm");*/
#endif

 filePtr = fopen("query_f", "w");  /* Creates the file */
 if (filePtr == NULL)
     printf("%s\n", "Error opening file!..");
 else {
    temp = abdlque_first_node;
    while(temp){
       strcpy(temptext, temp->text);
       strcat (temptext, "\n");
```

```
        fputs(temptext, filePtr);
        temp = temp -> next;
      }
    fclose(filePtr);
  }

#ifdef EnExFlag
 /*printf("%s\n", "Exit input_to_rtm");*/
#endif

 return;

}/* end of input_to_rtm */




void convert_delete_statement()
{
 /* will be implemented later ! */
}

void convert_insert_statement()
{
 /* will be implemented later ! */
}

void convert_if_statement()
{
 /* will be implemented later ! */
}

void convert_readInput_statement()
{
 /* will be implemented later ! */
}
```

APPENDIX I - DML COMPILER MAIN FILE

```
/*****************************************************************************

file name: dml_compiler.c

purpose :  This file serves as the main program for the dml_compiler. It

           assigns the input query file received from the user interface to

           "dmlin" which is the name of the file the scanner reads. It also

           calls the routine "dmlparse()" which is the parser.

*****************************************************************************/


#include <stdio.h>
#include "licommdata.h"
#include "ool_lildcl.h"


void dml_compiler()
 {
  extern FILE *dmlin;

  dmlin = ool_info_ptr->oi_file.fi_fid;


  printf("inside the dml_compiler()....\n");

  if (dmlparse() == 0)
    {
     printf("Successfully parsed!!\n");

    }
   else
    {
     printf( "Unsuccessfully parsed!!\n");
     exit(1);
    }
 }
```

APPENDIX J- THE INSTRUCTION MANUAL FOR THE OBJECT-ORIENTED DATA MANIPULATION LANGUAGE COMPILER(OODML)

J.1 Overview

The OODML compiler is based on a scanner produced with the program lex and a parser produced with the program yacc. Lex takes a set of descriptions of possible tokens and produces a C routine, which we call a scanner, that can identify those tokens. Our lex specification "dml_lex.l" contains sixty-five tokens. Yacc takes a concise description of a grammar and produces a C routine, which we call a parser, that can parse that grammar. Both lex and yacc are standard UNIX utilities and we used their original AT&T versions for our compiler.

Since the yacc parser is a higher level routine, it calls the lex scanner whenever it needs a token from the input query. The scanner then reads in the input of characters, until it finds a token of interest to the parser. The scanner only returns a token code.

In our implementation, the scanner does not bother the parser for comments and whitespaces. All the token codes were defined by yacc in order to agree with lex. Token zero signals for the logical end of the input.

We described the actual grammar as a set of production rules. Each rule consists of a single name on the left-hand side of the ":" operator, a list of symbols and action codes on the right-hand side, and a semicolon indicating the end of the rule. Our yacc specification "dml_yacc.y" contains eighty-four production rules. By default, the first rule is the highest-level rule ,i.e., the parser attempts to find a list of tokens which match this initial rule.

In order to fill the intermediate language table, the parser traces specific sequences of tokens in the production rules while it is parsing the query. As soon as the parser has enough tokens to recognize a structure for the intermediate table then it calls the related function declared in "intmed_tabl.c" to perform appropriate action.

After a query is successfully parsed, the execution passes to the function *query_constructor()* declared in "dml_comp.c" to make the necessary conversion for the kernel database.

J.2 Architectural Design

The OODML compiler conceptual model is a context diagram as shown in Figure j.1. The user interface of our system provides two options, either to choose any existing query or to write a new query from the terminal. If the user writes a queryname, the existing query is forwarded to the compiler with the query input file, then the OODML compiler processes the query and outputs either the pseudocode or an error message.

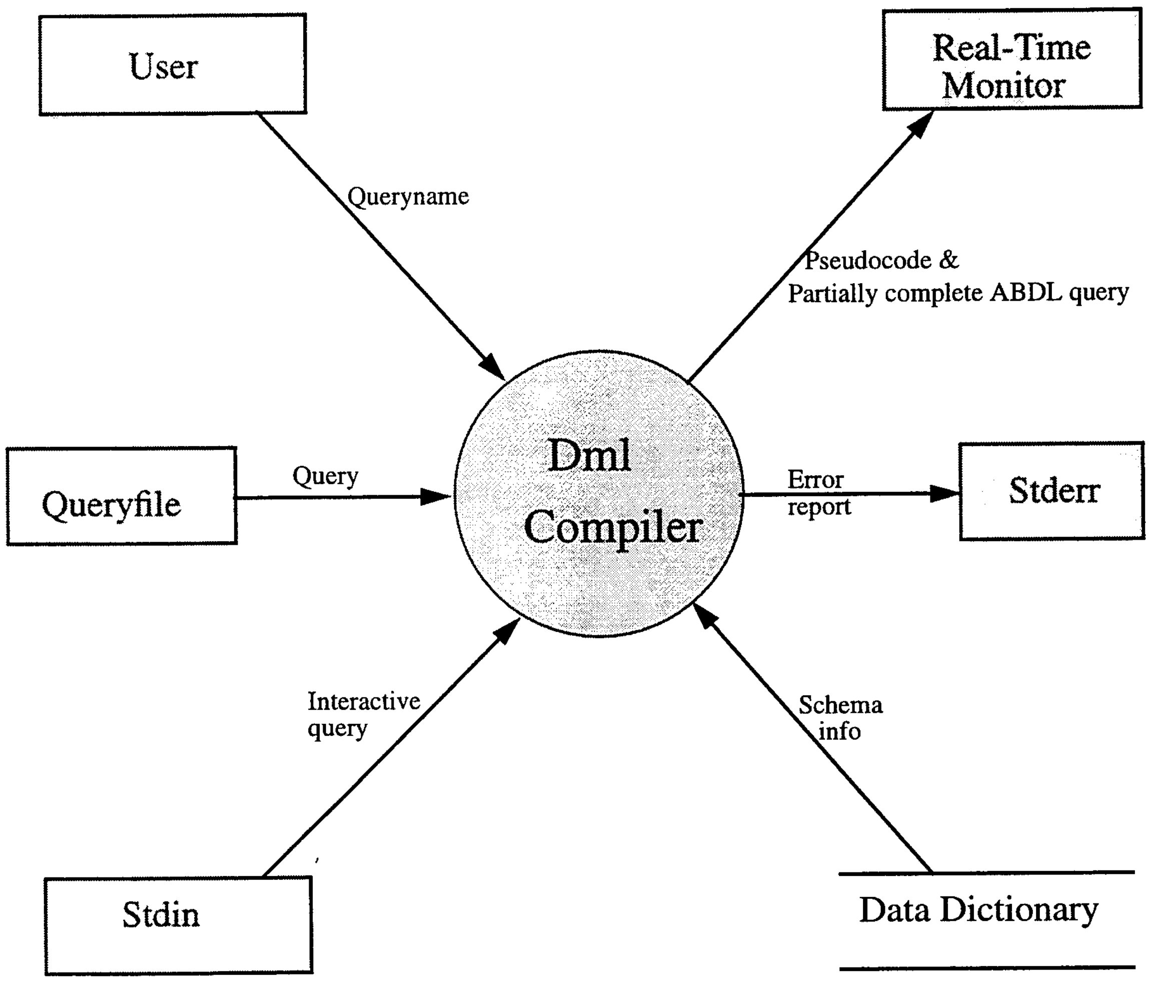


Figure j.1. Top-Level **Dml Compiler** Dataflow Diagram

The detailed elaboration of this context diagram is shown in Figure j.2.

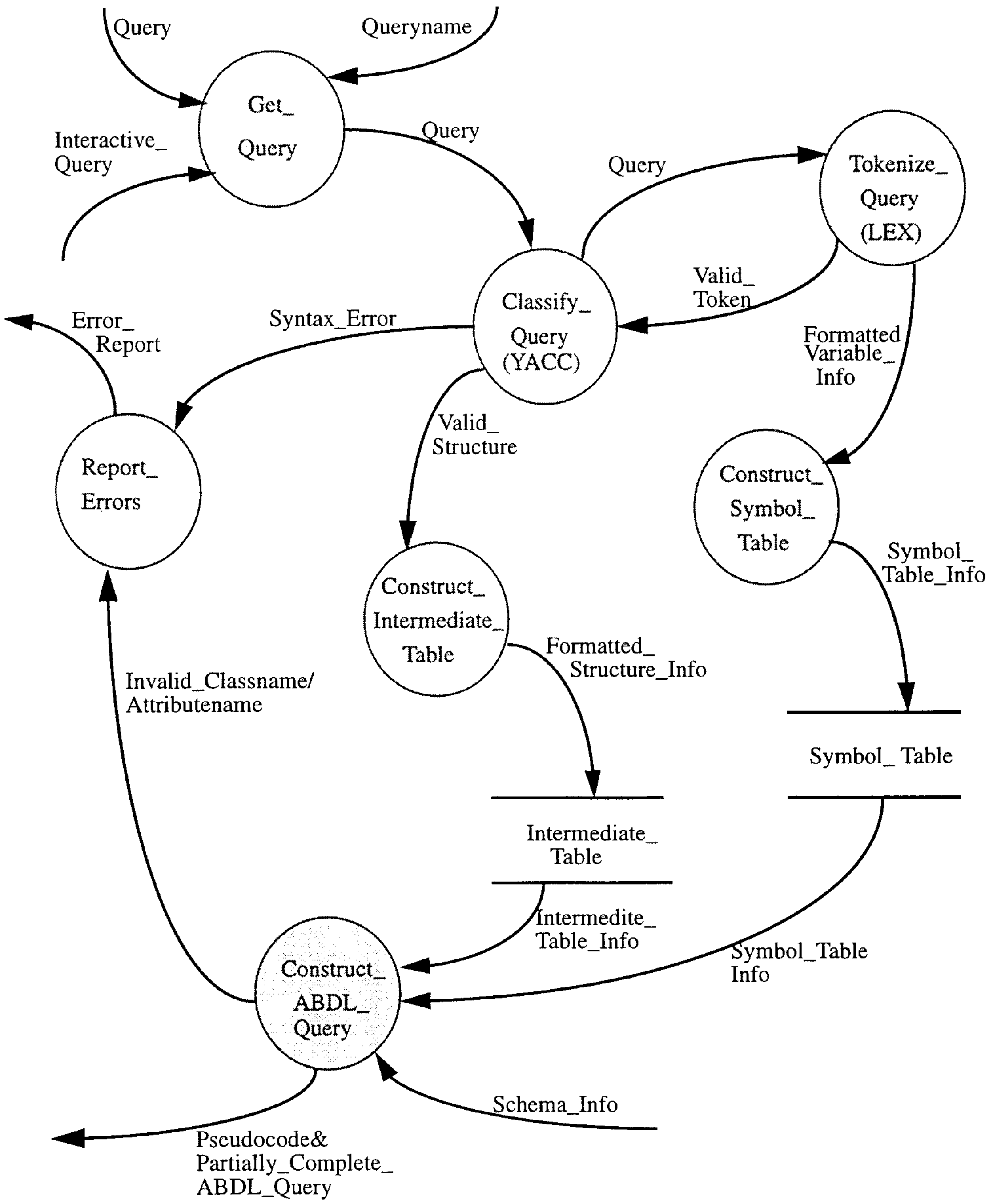


FIGURE j.2. Second-Level **Dml Compiler** Dataflow Diagram

The process "Get_Query" assigns the input queryfile received from the user interface to *dmlin* which is the name of the file the scanner reads, and calls the routine *dml-parse()* which is the parser generated by yacc.

The process “Get_Query” was implemented as the main() and declared in “dml_compiler.c”. The parser as a higher level routine calls *dmllex()* which is a scanner generated by lex whenever it needs a token. While the lex scanner is tokenizing each queryline under the supervision of the parser, if it encounters a type declaration then it calls the process “Construct_Symbol_Table”, which in turn, fills in the symbol table. The scanner sends all valid tokens to the parser and invalid tokens to the process “Report_Errors”. The process “Construct_Symbol_Table” was implemented as a function named *symtablook()* inside the lex specification “dml_lex.l”. The parser accepts all valid tokens and tries to match them with the defined grammar rules. While the parser is parsing the query, if it finds a structure of interest to the intermediate table, it calls the process “Construct_Intermediate_Table” with this structure, which in turn, fills in the intermediate table. This process was implemented as a series of functions, one for each structure, inside “intmed_tabl.c”. Function prototypes can be found in “intmed_tabl.h”. When the parser gets the last token from the scanner, either it will have returned “unsuccessfully parsed” message with the syntax error or it outputs a “successfully parsed” message and calls the process “Construct_ABDL_Query”.

“Construct_ABDL_Query” is somewhat the heart of the OODML compiler. This process takes all necessary information from the symbol table, intermediate language table, and data dictionary and constructs the output called the pseudocode and partially complete ABDL query. As shown in figure j.3, it checks the operation code for each structure in the intermediate table and selects the related branch for conversion. The process “Construct_ABDL_Query” was implemented as the main function for the *query_constructor()* and the conversion functions. The function prototypes can be found in “dml_comp.h”.

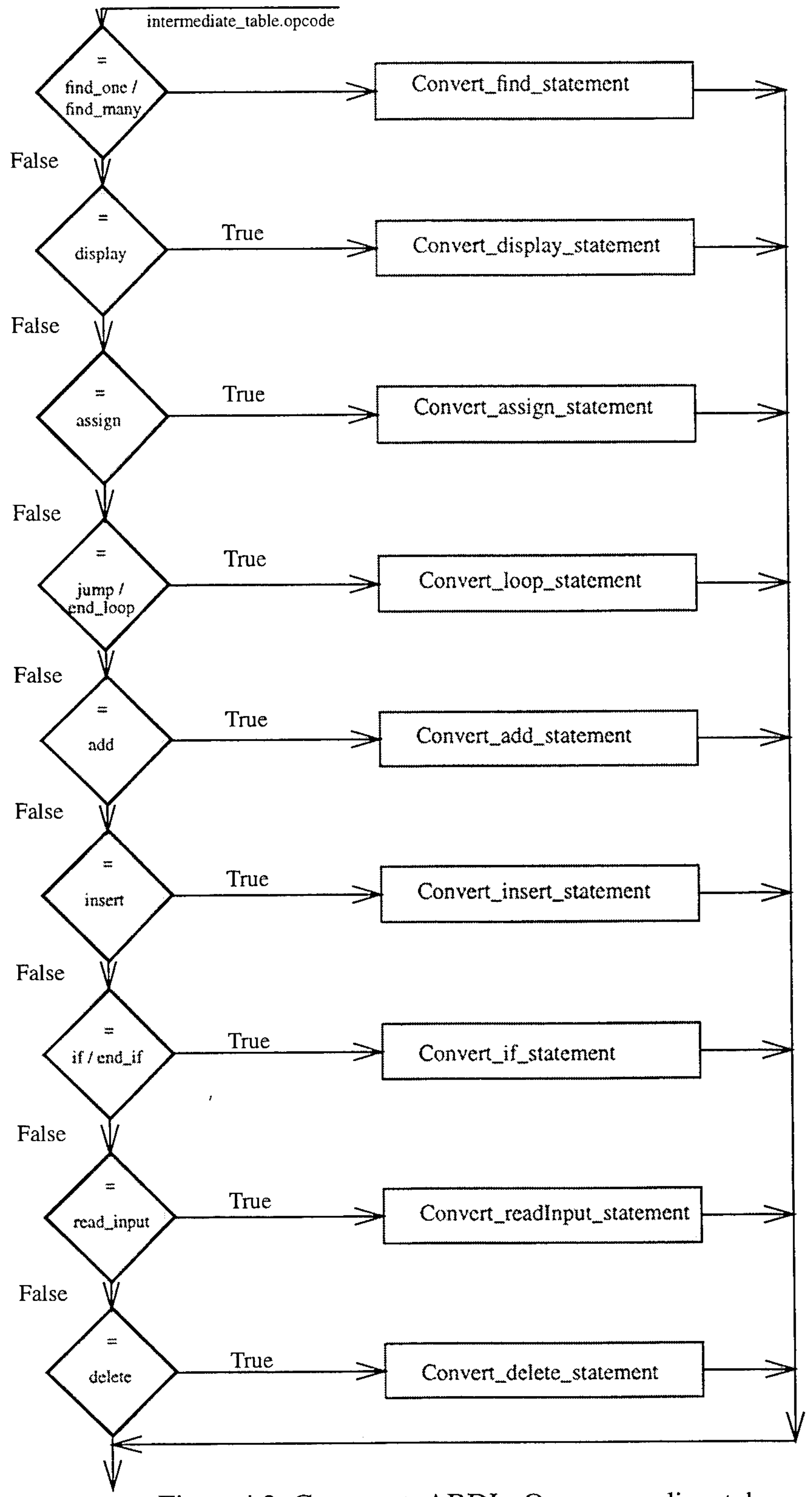


Figure j.3. Construct_ABDL_Query as a dispatcher

```
# file: Makefile (Obj/Lil)
# path:  db3 /usr/work/mdbs/rich/CNTRL/TI/LangIF/src/Obj/Kms/dml_comp/Makefile
#
# Insert names of sources here
SRCS= dml_compiler.c dml_comp.c intmed_tabl.c dml.tab.c lex.dml.c $(ALC)alloc.c
# Insert names of object files here
OBJECTS= dml_compiler.o dml_comp.o intmed_tabl.o dml.tab.o lex.dml.o $(ALC)alloc.o
# Insert names of include files here
INCLUDE= ../../../include
INCLUDES= $(INCLUDE)/licommdata.h $(INCLUDE)/ool.h $(INCLUDE)/ool_lildcl.h $(INCLUDE)/
ooldcl.h $(INCLUDE)/dml_comp.h $(INCLUDE)/intmed_tabl.h $(INCLUDE)/symtabl.h
#flags.def
ALC= ../Alloc/
CC= cc
#CFLAGS= -g -DEnExFlag -I$(INCLUDE)
CFLAGS= -O -I$(INCLUDE)
LPR= lpr
LPRFLAGS= -p
LIBS= -ll
all: $(INCLUDES) $(OBJECTS)
archive: $(SRCS)
               ci -u $(SRCS)
clean:
               -ci -q $(SRCS)
               -rm -f $(OBJECTS)
print: $(SRCS)
               $(LPR) $(LPRFLAGS) $(SRCS)
dml.tab.o: dml_yacc.y
               yacc -d dml_yacc.y
               sed -f yy-sed y.tab.c > dml.tab.c
               sed -f yy-sed y.tab.h > dml.tab.h
               -rm y.tab.c
               -rm y.tab.h
               cc $(CFLAGS) -c dml.tab.c
lex.dml.o: dml_lex.l
               lex dml_lex.l
               sed -f yy-lsed lex.yy.c > lex.dml.c
               -rm lex.yy.c
               cc $(CFLAGS) -c lex.dml.c
intmed_tabl.o:
               cp $(INCLUDE)/intmed_tabl.h .
               cc $(CFLAGS) -c intmed_tabl.c
dml_comp.o:
               cp $(INCLUDE)/ool_lildcl.h .
               cp $(INCLUDE)/licommdata.h .
               cp $(INCLUDE)/dml_comp.h .
               cc $(CFLAGS) -c dml_comp.c
               -rm ool_lildcl.h licommdata.h dml_comp.h
dml_compiler.o:
               cc $(CFLAGS) -c dml_compiler.c
#variable.o:
#              cp $(INCLUDE)/licommdata.h .
#              cp $(INCLUDE)/ddl_functions.h .
#              cp $(INCLUDE)/ool_lildcl.h .
#              cc $(CFLAGS) -c variable.c
#HFILES= licommdata.h
#dict_functions.o:
#              cc $(CFLAGS) -c dict_functions.c
#              -rm $(HFILES)
$(OBJECTS): $(INCLUDES)
```

Figure j.4. The KMS Subdirectory Makefile

There is no specific function written for the process "Error_Report", but all possible error messages are sent to the *stderr* using the C's *fprintf()* function.

Since the Object-Oriented Data Definition Language Group also used the UNIX tool *lex* and *yacc* for the OODDL Compiler, we renamed all functions generated by *lex* and *yacc* using UNIX stream editor *sed* to prevent the possible overwrite problem. We created two files "*yy-lsed*" and "*yy-sed*" for renaming the functions with the new prefix "*dml*". After we ran lex and yacc, the following commands edited the generated scanner and parser.

```
sed -f yy-lsed lex.yy.c > lex.dml.c
sed -f yy-sed y.tab.c > dml.tab.c
sed -f yy-sed y.tab.h > dml.tab.hJ
```

We put these rules in our "*Makefile*"[see Figure j.4]. So the above commands are executed automatically for each compilation process.

J.3 Incorpation of O-ODML Compiler into the Existing System

The Multi-model/Multi-lingual Database System(M^{2}DBMS) was designed to support many data languages. In order to support these data languages, the M^{2}DBMS provides a seperate user interface for each language. All of the user interfaces have identical control flows and structures. As shown in Figure j.5, the overall user interface structure consists of the Language Interface Layer(LIL), Kernel Mapping System(KMS), Kernel Formatting System(KFS), and Kernel Controller(KC) modules.

The LIL routes the user's input query written in the User Data Language(UDL) to KMS. The KMS translates this input query into equivalent Kernel Data Language(KDL) transactions. This process is called the data-language translation. The KMS routes the KDL transactions to the KC which in turn sends the KDL transactions to the Kernel DatabaseSystem(KDS) for execution. The KC routes any answer or response to the KFS for the KDM-to-UDM transformation. Once the transformation is complete, the KFS routes it to the LIL for the final relay to the user in user's data model/language form.

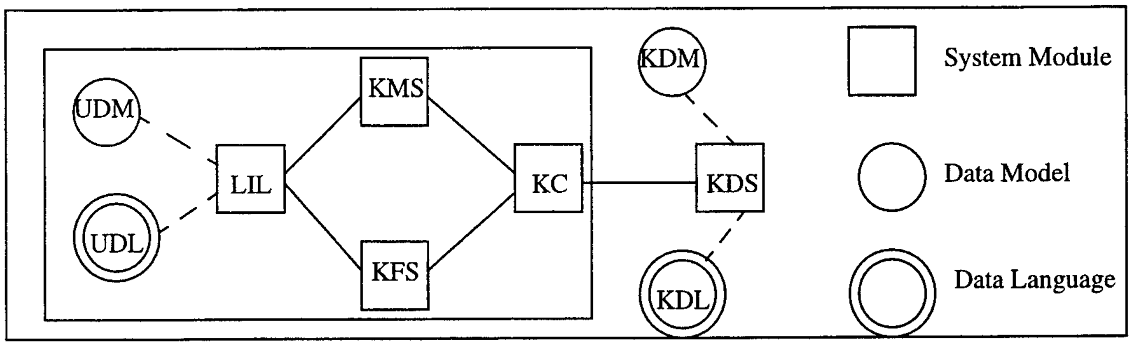


FIGURE j.5 A User Interface of M^{2}DBMS

The actual placement of all O-ODML Compiler component in such a user interface is within the KMS as shown in Figure j.6.

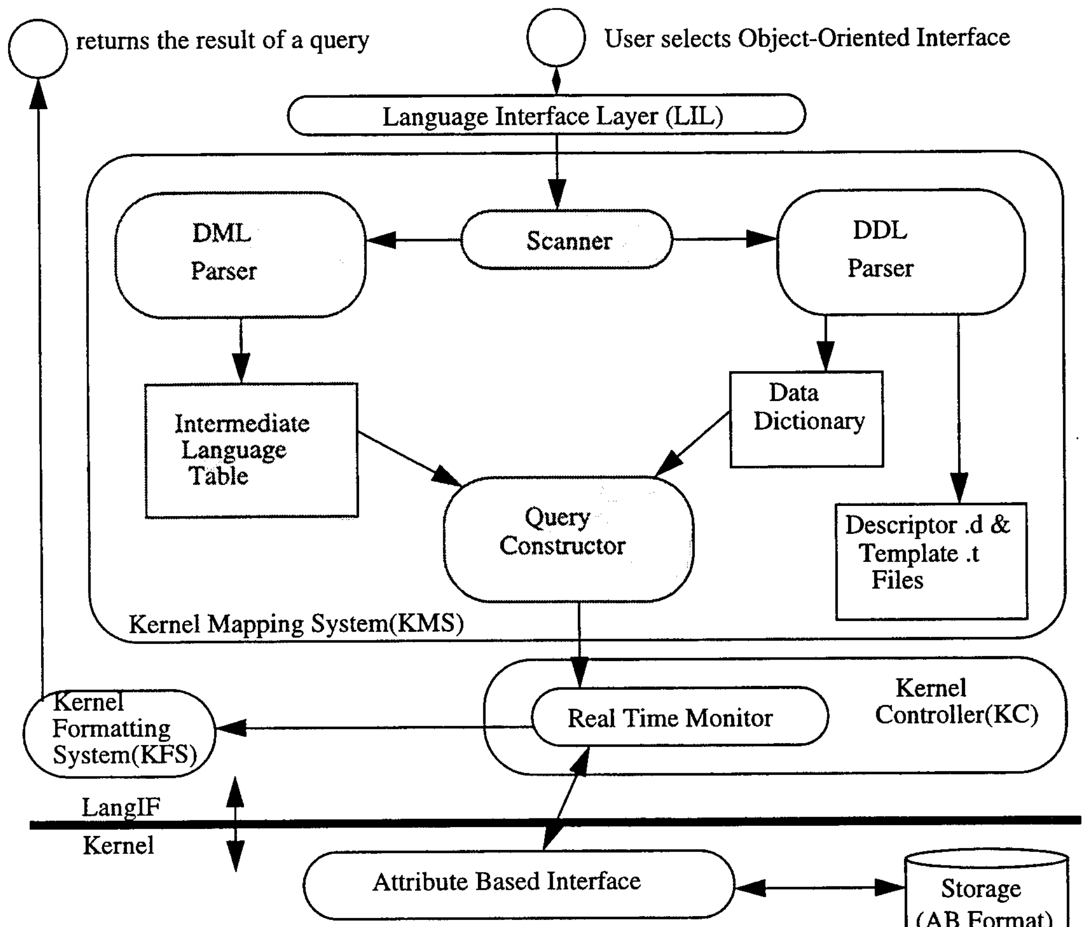

returns the result of a query
User selects Object-Oriented Interface
Language Interface Layer (LIL)
DML Parser
Scanner
DDL Parser
Intermediate Language Table
Data Dictionary
Query Constructor
Descriptor .d & Template .t Files
Kernel Mapping System(KMS)
Kernel Formatting System(KFS)
Real Time Monitor
Kernel Controller(KC)
LangIF
Kernel
Attribute Based Interface
Storage (AB Format)

FIGURE j.6 O-ODML Compiler Component Placement

As depicted in Figure j.6, the O-ODML Compiler consists of four subroutines: a scanner, a parser, the intermediate language table construction, and a query constructor.

J.4 Where To Find the OODML Compiler Files

All the C files for OODML Compiler can be found under the directory **greg/CNTRL/TI/langIF/src/Obj/Dml** (See Figure j.7)

Makefile	dml_lex.l	lex.dml.c	yy-lsed
dml.tab.c	dml_yacc.y	intmed_tabl.c	yy-sed
dml.tab.h	dml_compiler.c	dml_comp.c	

FIGURE j.7. Compiler files under the Dml directory

All the header files for function prototypes can be found under the directory **greg/CNTRL/TI/LangIF/include** (See Figure j.8)

dml_comp.h	intmed_tabl.h	symtabl.h

FIGURE j.8. Header files under the Include directory

J.5 How To Modify and Compile the OODML Compiler Files

Since the OODML Compiler is a part of the entire system, whenever a modification is done the whole system needs to be recompiled for a new executable. The System has a general “Makefile” for compiling the entire system. But there is also a specific “Makefile” for the Dml directory. Thus, after a modification in the OODML compiler, a new version can be compiled locally under the same directory. Modification for the header files can be done under the directory greg/CNTRL/TI/LangIF/include. To compile the entire system:

1. Change the directory to the mdbs/greg/CNTRL(**%cd ../../../../..**)
2. Remove the old executable(**%rm ti.exe**)
3. Change the directory to the mdbs/greg/CNTRL/TI(**%cd TI**)
4. Use Makefile to compile entire system(**%mk**)
5. Check for the compile errors(**%more make_result**)
6. If there is no error then the system is ready to run(**%start**) otherwise fix the error and repeat the step 4 and 5.

APPENDIX K- TRANSLATION ALGORITHM

Algorithm for Translating the Intermediate Language Table to Executable Pseudocode for the Real Time Monitor

This algorithm covers only the query constructor phase of the DML compiler. The original version was used to break down the problem into solvable units. It had to be updated at the completion of the query constructor to account for changes in the actual algorithm and new problems faced in interfacing with the ABDM.

This algorithm includes some symbology that is not from a programing language or may not be readily understood. Therefore the following explanations are given for clarification:

- On the rule and sub rule title line proceeding the "-" is the name of the corresponding function.
- The "##" is used to indicate a variable definition that is relevant to that function.
- Single quotes are used around variable names while double quotes are used around literals.
- The "*" is used to document limitations of a function.

Rule 1 - Conversion process initiation

tempset - character string that is used for compiler generated variables as they are needed for single references, when breaking down an O-O query into a series of AB queries.

tempref - character string that is used for compiler generated variables as they are needed for sets, when breaking down an O-O query into a series of AB queries.

1) Set up declarations of obj_ref and obj_set then place in the symbol table (accomplished by file "dml_lex.l". Additional declarations will be inserted into the symbol table, by the query constructor as needed, and once the query is completely translated the contents of the symbol table will be written to the linked list.

2) Create a linked list data dictionary from the original text version and return the pointer to the new dictionary.

3) Get the pointer to the first node (column and row) of the intermediate language table.

4) Read Operation block, if the operation block equals

find_one/find_many	goto rule2	"convert_find_statement()"
jump	goto rule 3	"convert_loop_statement()"
assignment	goto rule 4	"convert_assign_statement()"
display	goto rule 5	"convert_display_statement()"
add	goto rule 7	"convert_add_statement()"

Rule 2 - convert_find_statement()

logical_opar - variable that indicates whether an "and" or "or" is the operator used

r - variable that indicates the type of logical statement (for further information see the function convert_logical_statement in the file dml_comp.c

1) Set the type classification of the result

A) If this is a find_one query the type of the result must be an obj_ref, else if it is a find_many the result must be an obj_set. Check to see if the Result block variable (id_a) is of the correct type:

a) if true, continue

b) if false, Error message such as "type mismatch query construction terminated"

C) Set the result variable of this query to be (id_a) and set its class name to be that of (id_b) i.e. The final retrieve should produce an OID that identifies a member of the subset indicated by the class name of variable (id_b)

D) Set up a while loop and continue to execute until the result block of the intermediate language table has a value of 0

E) Set up a branch statement that covers:

logical operators "or"	convert_logical_statement
"and"	convert_logical_statement

(logical operators must handle precedence also)

contain statement "contains"	convert_logical_statement
relational operators "=,<,>"	convert_relational_statement

F) Go back to the while loop

E) Check the substitution flag variable 'subsflag'

1) If it has been set then insert the substitution warning and variable node prior to the assignment of the current query

2) If it has not been set then do nothing

G) End of while loop

Rule 3 convert_loop_statement()

1) Check to see if the contents of intmed_ptr -> opar is "jump"

A) If so, check to see if intmed_ptr -> arg1 and arg2 are in the symbol table and if so assign them to the pointers 'symptr1' and 'symptr2' respectively

a) If either or both do not exist in the symbol table, display an error and exit

B) The class name of the index variable symptr1 -> clsname gets the class name of the range variable symptr2 -> clsname

2) Allocate an abdlque_node

```
    A) Check to see if the contents of intmed_ptr -> opar is “jump”
        a) If so, Stringcopy “$” into ‘temptext’
            I) String concatenate intmed_ptr -> arg1 onto ‘temptext’
            II) String concatenate “,” onto ‘temptext’
            III) String concatenate intmed_ptr -> arg2 onto ‘temptext’
        b) If not, Stringcopy “!” onto ‘temptext’
            I) Add the node to the linked list (following linked list procedures)
```

Rule 4 Assignment (update)

```
1) Set the variable “a” equal to zero
2) Stringcopy “oid” into ‘temp_oid’
3) Copy the present global variable for arguement1 into ‘attr_arr’ by breaking arguement1
   up at the delimiters “.” (see step 3AaII1B in sub rule 5 convert_contains_statement for
   a step by step break down of this algo)
4) Take the tokenized results of step 3 (‘path_expr[0])’ and check to see if it is in the symbol table
   and if so, assign the pointer ‘symptr’ to it
    A) Set the variable “s” equal to zero and the variable ‘rf’ equal to one
    B) Stringcopy symptr -> clsname into ‘reftabl_arr[s]’
    C) Use a do while loop, (condition at the bottom of loop first entry mandatory)
        a) Check to see if ‘reftabl_arr[s]’ is in the data dictionary
            I) Check to see if the ‘path_expr[rf]’ is in the current class and if so assign the pointer
               ‘attr_ptr’ to it
                1) If so, check to see if the contents of attr_ptr -> dict_ref_type is the empty string
    A) If so, then call construct_update() and break
  2) Else, check to see if the contents of attr_ptr -> dict_ref_type is the string "ref"
     A) If so, set the 'subsflag' equal to one
     B) Call construct_retrieve_by()
     C) Increment the 'rf' variable
     D) Use a do while loop, (condition at the bottom of loop first entry mandatory)
        I) Stringcopy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
        II) Check to see if 'reftabl_arr[s]' is in the data dictionary
           1) Check to see if the 'path_expr[rf]' is in the current class and if so assign the
              pointer 'attr_ptr' to it
              A) If so, check to see if the contents of attr_ptr -> dict_ref_type is the empty
                 string
                 a) If so, then call construct_update() and break (This break gets you out
                    of the inner loop)
              B) Else, check to see if the contents of attr_ptr -> dict_ref_type is the string
                 "ref"
                 a) If so, set the 'subsflag' equal to one
                 b) Call construct_retrieve_by()
                 c) Increment the 'rf' variable
              C) Else, check the current class for inherit {using the check_inherit()} and
                 if so, assign the pointer attr_ptr to it
     E) End of inner do while loop, condition is that attr_ptr must exist
  3) Break, (This break gets you out of the outer loop)
```

II) Else, check the current class for inherit {using the check_inherit()} and if so, assign the pointer 'attr_ptr' to it

1) Stringcopy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'

D) End of outer do while loop, condition is that 'attr_ptr' must exist

<u>Rule 5 convert_display_statement()</u>

* note - This function has a limitation of six arguments.

1) Stringcopy "oid" into the string 'attrname'

2) Set the value of the variable 'b' equal to zero

3) Use a while loop with the condition that the contents of intmed_ptr -> opar equal the string "display"

A) Set the value of variables 'a' and 'k' equal to zero

B) Copy the present global variable for arguement1('arg1') into 'display_arr' by breaking up arguement1 at the delimiters "." (See step 3AaII1B, in sub rule 5 convert_contains_statement() for a step by step break down of this algo)

C) Check to see if intmed_ptr -> arg2 exists

a) If so, increment the variable 'b'

b) Set the value of the variables 'a' and 'k' equal to zero

c) Copy the present global variable for arguement1('arg1') into 'display_arr' by breaking up arguement1 at the delimiters "." (See step 3AaII1B, in sub rule 5 convert_contains_statement() for a step by step break down of this algo)

D) Increment the variable 'b'

E) Check to see if the contents of intmed_ptr -> result equals one

```
        A) If so, now move the intmed_ptr to the next node
      F) Else, break
  4) Set the value of the variable 's' equal to zero
  5) Set the value of the variable 'subsflag' equal to zero
  6) Check to see if the tokenized variables 'display_arr' are in the symbol table and if so copy the
     classname into 'reftabl_arr', else display an error
  7) Set the value of the variable 'rf' equal to one
  8) Use a do while loop, (condition at the bottom of loop first entry mandatory)
     A) Check to see if 'reftabl_arr[s]' is in the data dictionary
        a) If so, Check to see if the 'display_arr[i][rf]' is in the current class and if so assign the
           pointer 'attr_ptr' to it
           I) If so, check to see if the contents of attr_ptr -> dict_ref_type is the empty string
              1) IF so, Stringcopy the contents of 'display_arr[i][rf]' into 'first_attr'
                 A) Stringcopy the contents of 'first_attr' into 'attrlist'
                 B) Set up a for loop to check and see if the following arguments are in the same
                    table ('u' is the index variable and is initially set at i+1, loop until u<b
                    a) Check to see if the 'display_arr[u][rf]' is in the current class and if so assign the
                       pointer 'attr_temp' to it
                       I) If so, check to see if the contents of attr_temp -> dict_ref_type is the empty
                          string (Yes answer means that the following attribute is in the same table)
                          1) If so, String concatenate "," into 'attrlist'
                             A) String concatenate the contents of 'display_arr[u][rf]' into 'attrlist'
                             B) Increment the variable 'i'
```

```
            2) Else, Break (not in the same table)
      C) Call construct_retrieve_display()
      D) Break
  2) Else, check to see if the contents of attr_ptr -> dict_ref_type is "ref"
      A) If so, set the 'subsflag' variable equal to one
          a) Call construct_retrieve_by()
          b) Increment the variable 'rf'
          c) Use a do while loop, (condition at the bottom of loop first entry mandatory)
              I) Stringcopy the value of attr_ptr -> dict_ref_type  into 'reftabl_arr[s]'
              II) Check to see if 'reftabl_arr[s]' is in the data dictionary
                  1) Check to see if the 'display_arr[i][rf]' is in the current class and if so
                     assign the pointer 'attr_ptr' to it
                     A) If so, check to see if the contents of attr_ptr -> dict_ref_type is the
                        empty string
                        a) IF so, Stringcopy the contents of 'display_arr[i][rf]' into 'first_attr'
                            I) Stringcopy the contents of 'first_attr' into 'attrlist'
                            II) Set up a for loop to check and see if the following arguments
                                are in the same table ('u' is the index variable and is initially
                                set at i+1, loop until u<b)
                                1) Check to see if there presently exists an attribute and that it
                                   and the previous attribute have the same path expression
                                   A) If so, assign the pointer 'attr_temp' to 'display_arr[u][rf]'
                                      a) Check to see if the contents of attr_temp -> dict_ref_type
```

```
                    is the empty string
                    I) If so, String concatenate "," onto 'attrlist'
                    1) String concatenate 'display_arr[u][rf]' onto 'attrlist'
                      2) Increment the variable 'i'
                    II) Else, Break (path expression not the same)
                  B) Else, Break (current attribute is not in the same table
              III) Call construct_retrieve_display()
              IV) Break (found primitive value and called construct)
            b) Break  (gets out of do loop when primitive value found
          B) Else, check to see if the contents of attr_ptr -> dict_ref_type is "ref"
            a) If so, set the 'subsflag' variable equal to one
              I) Call construct_retrieve_by()
              II) Increment the variable 'rf'
           C) Else, check inherit for the second attribute (The second attribute
             was not in the table) and assign it to the pointer 'attr_ptr'
      d) End of while loop if the 'attr_ptr' exist loop back up
  3) Else, check inherit for the second attribute (The second attribute was not in the
    table) and assign it to the pointer 'attr_ptr
    A) If so, Stringcopy attr_ptr -> dict_ref_table into reftabl_arr[s]
9) End of while loop if the attr_ptr exist loop back up
```

<u>Rule 6 add convert_add_statement()</u>

```
1) Set the variable 'a' equal to zero
```

```
2) Copy the present global variable for arguement1 into 'path_expr[a][k]' by breaking
   arguement1 up at the delimiters "." (see step 3AaII1B in sub rule 5 convert_contains_statement
   for a step by step break down of this algo)
3) Take the tokenized results of step 3 ('path_expr[0])' and check to see if it is in the symbol table
   and if so, assign the pointer 'symptr' to it
4) Set the variable 's' equal to zero
5) Stringcopy symptr -> clsname into 'reftabl_arr[s]'
6) Check to see if 'reftabl_arr[s]' is in the data dictionary
   A) Check to see if 'a' is greater than zero (if so it is a store add)
      a) Check to see if the 'path_expr[1]' is in the current class and if so assign the pointer
         'attr_ptr' to it
         I) If so, check to see if the contents of intmed_ptr -> arg2 are in the symbol table and
            assign it to the pointer 'symptr'
            1) If so, increment the variable 's'
               A) Stringcopy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
               B) Check to see if 'reftabl_arr[s]' is in the data dictionary
                  a) If so, Call construct_add()
   B) Else, check the current class for cover {using the check_cover()} and if so, assign the
      pointer 'attr_ptr' to it
      a) Increment the variable 's'
         I) Stringcopy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
         II) Check to see if the contents of intmed_ptr -> arg2 are in the symbol table and
             assign it to the pointer 'symptr'
```

```
        1) If not error
        2) Else, (if so)
           A) Check to see if 'reftabl_arr[s]' is in the data dictionary
              a) If so, Call construct_add()
```

<u>Sub Rule 1 - convert_rel_expr()</u>

```
1) Initialize the a temporary array ('attr_arr[]') to an empty array
2) Copy the present global variable for arguement1('arg1') into 'attr_arr[]' by breaking up
   arguement1 at the delimiters "." (See step 3AaII1B, in sub rule 5 convert_contains_statement()
   for a step by step break down of this algo)
3) Check to see if the current path expression for this class is in the data dictionary and
   set the pointer on this class
   A) If so, check to see if the current attribute 'attr_arr' is in the class and set the 'attr_ptr' to
      point to this attribute column of the class
      a) If so, check to see if the 'dict_ref_type' column is empty
         I) If so, check to see if argument 2 is in the symbol table
            1) If so, set the substitution flag equal to one
            2) Call construct_retrieve_gen()
            3) Return the 'objset_arr[p-1]'
      b) Else, check to see if the 'dict_ref_type' column contains the key word "ref"
         I) If so, increment the 's' and 'a' variable counters
         II) String copy the value of attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
         III) String copy the value of construct_retreive_ref() into the variable 'tempvar'
```

```
    IV) Decrement the 's' and 'a' variable counters
    V) Set the substitution flag equal to one
    VI) Call construct_retrieve_gen()
    VII) Return the 'objset_arr[p-1]'
B) If 'attr_arr' is not in the class check all the 'dict_ref_type' column for all the attributes in
   that class and return the attribute pointer to the attribute row with an "inherit"
   {check_inherit()}
   a) Initialize the inheritance counter to zero
   b) While the attr_ptr = check_inherit() LOOP
      I) Increment the 'inherit_cnt' and the 's' variables
      II) String copy the value of attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
      III) Check to see if the path expression for the class name in 'reftabl_arr[s]' is in the data
           dictionary
           1) If so, check to see if the current attribute 'attr_arr' is in the class and set the
              'attr_ptr' to point to this attribute column of the class
              A) If so, check to see if the 'dict_ref_type' column is empty
                 a) If so, Call construct_retrieve_gen()
                 b) String copy an empty string into 'reftabl_arr[s]'
                 c) 's' gets the value of 's' minus the value of 'inherit_cnt'
              B) If so, and the 'dict_ref_type' column contains "ref" then
                 a) Increment the 's' variable
                 b) String copy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
                 c) Increment the 'a' variable
```

d) String copy the char * returned from construct_retrieve_ref() {by calling construct_retrieve_ref() inside of a strcpy} into 'tem_val'

e) String copy an empty string into 'reftabl_arr[s]'

f) Decrement 's' and 'a' variables

g) Set the value of 'subsflag' equal to one

h) Call construct_retrieve_gen()

i) 's' gets the value of 's' minus the value of 'inherit_cnt'

j) Return the 'objset_arr[p-1]'

C) Else, 'attr_ptr[0]' is not a valid attribute name

4) Else, 'reftabl_arr[s]' does not exist in the dictionary

Sub Rule 2 convert_logical_statement()

1) Initialize the 'left' and 'right' variables to zero

2) Decrement the index (variable 'r' from convert_find_statement)

3) Branch to the correct case statement according to the index value

4) Call construct_retrieve_logical() with the new values of 'left' and 'right'

Sub Rule 3 - construct_retrieve_ref()

1) Check to see if the path expression for the class name in 'reftabl_arr[s]" is in the data dictionary

A) If so, check to see if the current attribute 'attr_arr' is in the class and set the 'attr_ptr' to point to this attribute column of the class

a) If so, check to see if the 'dict_ref_type' column is empty

I) If so, Call construct_retrieve_gen()

II) Return the 'objset_arr[p-1]'

b) If so, and the dict_ref_type column contains "ref" then

I) Increment the 's' and 'a' variables

II) String copy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'

III) String copy the char * returned from construct_retrieve_ref() {by calling construct_retrieve_ref() inside of a strcpy} into 'tem_var'

IV) Decrement the 's' and 'a' variables

V) Set the value of 'subsflag' equal to 1

VI) Call construct_retrieve_gen()

VII) Return the 'objset_arr[p-1]'

c) If 'attr_arr' is not in the class check all the dict_ref_type column for all the attributes in that class and return the attribute pointer to the attribute row with an "inherit" {check_inherit()}

I) Increment the 's' variable

II) String copy attr_ptr -> dict_ref_table into 'reftabl_arr[s]'

III) String copy the char * returned from construct_retrieve_ref() {by calling construct_retrieve_ref() inside of a strcpy} into 'tem_var'

IV) Decrement the 's' variable

V) Return 'temp_var'

<u>Sub Rule 4 construct_retrieve_gen()</u>

temptext - string used to store the text of a query (the string is a member of the struct/node)

```
## ref_temp - string used solely to store and manipulate strings prior to concatenation onto
      temptext
1) Allocate an abdlque_node
2) Stringcopy an "&" into string 'temptext'
3) String concatenate arg1 onto 'temptext' and copy 'temptext' in abdlque_current_node->text
4) Add the node to the linked list (following linked list procedures)
5) If 'subsflag' is greater than 0 than
   A) Allocate an abdlque_node
   B) Stringcopy a "~" into string 'temptext'
   C) String concatenate arg2 onto 'temptext' and copy 'temptext' in
      abdlque_current_node -> text
   D) Add the node to the linked list (following linked list procedures)
6) Allocate an abdlque_node
7) Stringcopy a "[RETRIEVE ((TEMP =" into string 'temptext'
8) Stringcopy a "reftabl" into string 'ref_temp'
9) String concatenate the results from function call convertoclassname(ref_temp) onto 'temptext'
10) String concatenate ") and (" onto 'temptext'
11) Stringcopy a 'attrname' into string 'ref_temp'
12) String concatenate the results from function call convertoupper(ref_temp) onto 'temptext'
13) String concatenate 'relexpr' onto 'temptext'
14) String concatenate arg2 onto 'temptext'
15) String concatenate "))(OID)] onto 'temptext' and copy 'temptext' in
      abdlque_current_node->text
```

16) Add the node to the linked list (following linked list procedures)

<u>Sub rule 5 convert_contains_statement()</u>

```
1) Check to see if arg1 is the string "it"
  A) If so, Check to see if the path expression for the class name in 'reftabl_arr[0]' is in the data
     dictionary
    a) If so, let 'attr_ptr' point to the attribute which is of relationship type "cover"
      I) Increment the variable 's'
      II) Stringcopy the contents of attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
      IV) Check to see if the path expression for the class name in 'reftabl_arr[s]' is in the data
          dictionary
        1) If so, Stringcopy "oid_" onto string 'temp1'
        2) Check to see if intmed_ptr -> arg2 is in the symbol table and assign it to the pointer
           'symtab_ptr'
          A) If NOT, then output a message "undeclared variable"
          B) If So,  String concatenate "symtab_ptr -> clsname" into string 'temp1'
          C) Check to see if 'temp1' is in the present class and assign its attribute pointer
             to 'attr_ptr'
            a) If so,  Stringcopy "OID_" onto string 'temp2'
            b) Stringcopy 'reftabl_arr[s-1]' onto string 'temptext'
            c) String concatenate the result of the converttoupper(temptext) onto the
               string 'temp2'
            d) Call convertoupper(temp1) to insure correct format
            e) Set the value of 'subsflag' equal to one
            f) Call construct_retrieve_by()
            g) Set the value of 'logopar' equal to two
            h) Call construct_retrieve_logical()
            i) Stringcopy an empty string into 'reftabl_arr[s]'
            j) Decrement the variable 's'
            k) Set the 'subsflag' equal to 0
            l) Return the 'objset_arr[p-1]'
2) Check to see if 'arg2' is the string "it"
  A) **** This type of covering will be implemented by follow on groups ****
3) Else, String copy the contents of intmed_ptr -> arg1 into the array 'attr_arr[0]'
   A) Check to see if 'reftabl_arr[s]' is in the data dictionary and if so set the pointer
      a) If so, check to see if 'attr_arr[0]' is in the present class, if so set the pointer 'attr_ptr' the
         attribute
         I) Check to see if the contents of attr_ptr -> dict_attr_type is "set_of"
            1) If so, increment the 's' variable
            2) Stringcopy the contents of attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
            3) Check to see if the contents of 'reftabl_arr[s]' is in the data dictionary
               A) If so, stringcopy "oid_" into 'temp1'
               B) Check to see if the contents of intmed_ptr -> arg2 is in the symbol table, if so
                  then have the pointer 'symtab_ptr' point at it
                  a) If so, string concatenate the contents of symtab_ptr -> clsname onto 'temp1'
                  b) Check to see if the concatenated contents of 'temp1' is one of the hidden
            attributes in the class, and if so, have the pointer 'attr_ptr' point to it
            I) If so, stringcopy "OID_" into the string 'temp2'
            II) Stringcopy the contents of 'reftabl_arr[s-1]' into 'temptext'
            III) Convert the contents of 'temptext' to upper case (converttouppper) and
               string concatenate it onto 'temp2'
            IV) Convert the contents of 'temp1' to upper case (converttouppper)
            V) Set the substitution flag equal to one
            VI) Call construct_retrieve_by()
            VII) Stringcopy an  empty string into 'reftabl_arr[s]'
            VIII) Decrement the 's' variable
             IX) Return 'objset_arr[p-1]'
II) Check to see if the contents of attr_ptr -> dict_attr_type is "inverse_of"
   1) If so, set the variables 'k' and 'l' equal to zero
      A) Find the string length of attr_ptr -> dict_ref_table and set the variable length
         equal to it
      B) Use a for loop to count from 0 to length(i) LOOP (Used to tokenize attribute
         names
         a) Check to see if the value of attr_ptr -> dict_ref_table[i] is "."
            I) If so, 'settype_arr[k][l]' gets the value "\0" (end of string)
               1) increment the variable 'k'
               2) Set the value of the variable 'l' equal to zero
            II) If not, 'settype_arr[k][l]' gets the value of attr_ptr -> dict_ref_table[i]
         b) 'settype_arr[k][l]' gets the value "\0" (end of string)
```

```
4) Check to see if 'settype_arr[0]' is in the data dictionary
  A) If so, check to see if 'settype_arr[1]' is in the current class and if so set the pointer
     'attr_ptr' to point at it
     a) Increment the variable 's'
     b) Stringcopy the contents of attr_ptr -> dict_ref_table into 'reftabl_arr[s]'
     c) Check to see if 'reftabl_arr[s]' is in the data dictionary
        I) If so, Stringcopy "oid_" into 'temp1'
        II) Check to see if intmed_ptr -> arg2 is in the symbol table and if so assign the pointer
            'symtab_ptr' to it (error message are appropriate here if not)
            1) If so, string concatenate the contents of symtab_ptr -> clsname onto 'temp1'
            2) Check to see if the contents of 'temp1' is in the current class and if so assign the
               pointer to it
               A) If so, stringcopy "OID_"  into 'temp2'
               B) Stringcopy the contents of 'reftabl_arr[s-1]' into 'temptext'
               C) Convert the contents of 'temptext' to uppercase and concatenate the results to
                  'temp2'
               D) Convert the contents of 'temp1'  to uppercase
               E) Set the value of the substitution to one
               F) Call Construct_retrieve_by()
               G) Stringcopy an empty string into 'reftabl_arr[s]'
               H) Decrement the variable 's'
               I) Return 'obset_arr[p-1]'
```

Sub rule 6 construct_retrieve_by

```
## temptext - string used to store the text of a query (the string is a member of the struct/node)
## ref_temp - string used solely to store and manipulate strings prior to concatenation onto
    temptext
1) Allocate an abdlque_node
2) Stringcopy an "&" into string 'temptext'
3) String concatenate arg1 onto 'temptext' and copy 'temptext' in abdlque_current_node->text
4) Add the node to the linked list (following linked list procedures)
5) Allocate an abdlque_node
6) If 'subsflag' is greater than 0 than
   A) Allocate an abdlque_node
   B) Stringcopy a "~" into string 'temptext'
   C) String concatenate arg2 onto 'temptext' and copy 'temptext' in
      abdlque_current_node->text
   D) Add the node to the linked list (following linked list procedures)
7) Allocate an abdlque_node
8) Stringcopy a "[RETRIEVE ((TEMP = ")" into string 'temptext'
9) Stringcopy a 'reftabl' into string 'ref_temp'
10) String concatenate the results from function call convertoclassname(ref_temp)
    onto 'temptext'
11) String concatenate ") and (" onto 'temptext'
12) Stringcopy 'attrname' into string 'ref_temp'
13) String concatenate the results from function call convertoupper(ref_temp) onto 'temptext'
```

14) String concatenate relexpr onto 'temptext'

15) String concatenate 'arg2' onto 'temptext'

16) String concatenate "))(" onto 'temptext'

17) Stringcopy targetlist into string 'ref_temp'

18) Convert ref_temp to uppercase and concatenate onto 'temptext'

19) String concatenate ")BY" onto 'temptext'

20) Convert 'sortvar' to uppercase and concatenate onto 'temptext'

21) String concatenate "]" onto temptext and copy 'temptext' in abdlque_current_node->text

22) Add the node to the linked list (following linked list procedures)

<u>Sub rule 7 construct_retrieve_logical()</u>

1) Allocate an abdlque_node

2) Stringcopy an "&" into string 'temptext'

3) String concatenate 'varname' into string 'temptext' and copy 'temptext' in abdlque_current_node->text

4) Check to see if there is a prior node and add the node according to linked list procedures

5) Allocate an abdlque_node

6) Create a case statement for 'logflag' (the logical operator)

7) Stringcopy one of the following into 'temptext':

   case(0) "+" pseudocode for union

   case(1) "*" pseudocode for cross product

   case(2) "^" pseudocode for get common (similar to cross product but used with covering)

8) String concatenate 'leftvar' into string 'temptext'

9) String concatenate "," into string 'temptext'

10) String concatenate 'rightvar into string 'temptext'

11) Stringcopy temptext into abdlque_current_node -> text

12) Add the node to the linked list (following linked list procedures)

Sub rule 8 construct_retrieve_display()

1) Allocate an abdlque_node

2) Stringcopy a "~" into string 'temptext'

3) String concatenate 'varname' onto 'temptext' and copy 'temptext' into
abdlque_current_node->text

4) Add the node to the linked list (following linked list procedures)

5) Allocate an abdlque_node

6) Stringcopy a "[ORETRIEVE ((TEMP = ")" into string 'temptext'

7) Stringcopy a 'reftabl' into string 'ref_temp'

8) String concatenate the results from function call convertoclassname(ref_temp)
onto 'temptext'

9) String concatenate ") and (OID=" onto 'temptext'

10) String concatenate 'varname' onto 'temptext'

11) String concatenate "))(" onto 'temptext'

12) String concatenate the results from function call convertoupper(targetlist) onto 'temptext'

13) String concatenate ")BY" onto 'temptext'

14) String concatenate the results from function call convertoupper(sortvar) onto 'temptext'

15) String concatenate "]" onto temptext and copy 'temptext' in abdlque_current_node->text

16) Add the node to the linked list (following linked list procedures)

<u>Sub rule 9 construct_update()</u>

1) Allocate an abdlque_node

2) Stringcopy an "~" into string 'temptext'

3) String concatenate 'oid_value' onto 'temptext' and copy 'temptext' in abdlque_current_node->text

4) Add the node to the linked list (following linked list procedures)

5) Allocate an abdlque_node

6) Stringcopy a "[UPDATE ((TEMP = ")" into string 'temptext'

7) Stringcopy a 'reftabl' into string 'ref_temp'

8) String concatenate the results from function call convertoclassname(ref_temp) onto 'temptext'

9) String concatenate ") and (OID=" onto 'temptext'

10) String concatenate 'oid_value' onto 'temptext'

11) String concatenate "))<" onto 'temptext'

12) Stringcopy 'attrname' into 'ref_temp'

13) String concatenate the results from function call convertoupper(ref_temp) onto 'temptext'

14) String concatenate "=" onto 'temptext'

15) String concatenate 'attrvalue' into 'ref_temp'

16) String concatenate ">]" onto temptext and copy 'temptext' in abdlque_current_node->text

17) Add the node to the linked list (following linked list procedures)

Sub rule 10 construct_add()

```
1) Allocate an abdlque_node
2) Stringcopy an "#" into string 'temptext'
3) String concatenate 'attr_arg1' onto 'temptext'
4) String concatenate "," onto 'temptext'
5) String concatenate 'attr_arg2' onto 'temptext'
6) Copy 'temptext' in  abdlque_current_node->text
7) Add the node to the linked list (following linked list procedures)
8) Allocate an abdlque_node
9) Stringcopy a "[INSERT (<TEMP = ")" into string 'temptext'
10) Stringcopy a 'reftabl' into string 'ref_temp'
11) String concatenate the results from function call convertoclassname(ref_temp)
    onto 'temptext'
12) String concatenate ">,<OID,?>,<" onto 'temptext'
13) Assign the reference of 'dict_attr_ptr' to 'dict_att_temp'
14) Assign the reference of dict_att_temp -> dict_attr_name to 'dict_att_temp'
15) String concatenate dict_att_temp -> dict_attr_name onto 'temptext'
16) String concatenate "," onto "temptext"
17) Check to see if the contents of arg1_cls_name is equal to the contents of
    dict_att_temp -> dict_ref_table
    A) If so, string concatenate 'attr_arg1' onto 'temptext'
    B) Else, string concatenate 'attr_arg2' onto 'temptext'
18) String concatenate ">,<" onto 'temptext'
```

19) Assign the reference of dict_att_temp -> dict_attr_name to 'dict_att_temp'

20) String concatenate dict_att_temp -> dict_attr_name onto 'temptext'

21) String concatenate "," onto 'temptext'

22) Check to see if the contents of arg1_cls_name is equal to the contents of

dict_att_temp -> dict_ref_table

A) If so, string concatenate 'attr_arg1' onto 'temptext'

B) Else, tring concatenate 'attr_arg2' onto 'temptext'

23) String concatenate ">)]" onto 'temptext' and copy 'temptext' in abdlque_current_node->text

24) Add the node to the linked list (following linked list procedures)

APPENDIX L- DML TO RTM PSEUDOCODE

This document describes the pseudocode developed to assist the RTM in processing queries.

Declarations:

The declaration of variables in the Real time monitor pseudocode will be accomplished by using the symbols '%' and '@'.

```
obj_ref  :   %           obj_ref  a      :  % a
obj_set  :    @          obj_set  b,c,d  :   @ b,c,d
```

Assignment:

Assignments in the Real time monitor pseudocode will be accomplished by using the '&' followed by the desired variable that is to accept the assignment. No other entries on that line. The next line will then have to be one of the five Attribute based functions.

```
 :=     :   &            a := find_One.....  :      &a
                                                 RETRIEVE [( ......)(OID)]
```

For Loop:

For loops are handled by using the '$' followed by the index variable (must of type obj_ref) a comma then the set variable (must be an obj_ref).

```
For Each  i in b          :        $ a, b
.....                              DELETE (TEMP=student) (OID=a)
End_Loop;                          $
```

End For Loop :

For loops are ended by using "!" by itself.

```
.....                              DELETE (TEMP=student) (OID=a)
End_Loop;                          !
```

Cross Product:

Cross products are included in the pseudocode to accommodate for the lack of functionality in the retrieve common provided in the ABDM. The cross product (or more correctly stated equijoin) symbol is '*' and is followed by two variables containing the sets to be joined.

```
&b
[RETRIEVE(TEMP=Name)(lname=Jones)(OID)]
```

```
&c
[RETRIEVE(TEMP=address)(city=monterey)(OID)]
&d
* b,c
```

Union :
Or is accomplished by using "+" followed by the variables to be unioned (separated by a comma)

```
+ sb, sc
```

Substitution Warning:
In order to simplify the job of the real time monitor a method was developed to indicate that a substitution is required in the next line of the ABDL query and what variable was to be involved. The '~' followed by the variable to be substituted in the next line, is how this is accomplished.

```
~ p, pa
```

Double Substitution Warning:
This fills the same requirements of the substitution warning with the exception that two variables will be substituted instead of only one. This occurs only in the ADD and Delete operations due to the manner in which sets are stored in the ABDM. The '#' followed by the variables to be substituted in the next line, is how this is accomplished.

```
# sa,sb
```

Get Common Warning:
This fills nearly the same requirements of the cross product but is specifically used with the cover relationship. The '^' followed by the variables to be used in the next line, is how this is accomplished. Moreover the constraints on this is that the first argument must be of type object set while the second arguement may be either an object set or object reference.

```
^ sa, sb
```

LIST OF REFERENCE

[1] Hsiao, David K., "Interoperable and Multidatabase Solutions for Heterogeneous Databases and Transactions", a speech delivered at ACM CSC '95, Nashville, Tennessee, March 1995.

[2] Ramirez, L. and Tan, R., M., "The Design and Implementation of a Compiler for the Object-Oriented Data Definition Language (O-ODDL Compiler)", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[3] Badgett, R.B., "The Design and Specification of an Object-Oriented Data Definition Language(O-ODDL)", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[4] Stephens, M.W., "Design and Specification of an Object-Oriented Data Manipulation Language(O-ODML)", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[5] Senocak, E., "The Design and Implementation of a Real-Time Monitor for the Execution of Compiled Object-Oriented Transactions (O-ODDL and O-ODML Monitor)", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[6] Lesk, M.E. and Schmidt, E., *Lex - A Lexical Analzer Generator*, Bell Laboratories, Murray Hill, New Jersey, July 1978.

[7] Johnson, S.C., *Yacc: Yet Another Compiler-Compiler*, Bell Laboratories, Murray Hill, New Jersey, July 1978.

[8] Levine, J.R., Mason, T., and Brown, d., lex & yacc, 2nd Edition, O'Reilly & Associates, Inc., Sebastopol, California, October 1992.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Code 013
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Dr Ted Lewis, Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr David K. Hsiao, Code CS/HS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr C. Thomas Wu, Code CS/KA 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Lt Carlos Barbosa. 1
   Rd. 1 Box 147
   Georgetown, DE. 19947

7. LtJg Aykut Kutlusan 2
   Hocaalizade Mah. Manolya Sok.
   Manolya Apt. D:4 16010
   Bursa, TURKEY

8. Ms Sharon Cain 2
   NAIC/SCDD
   4115 Hebble Creek Rd
   Wright Patterson AFB, OH 45433-5622

9. Ms Doris Mleczko 2
Code P22305
Weapons Division
Naval Air Warfare Center
Pt Mugu, CA 93042-5001